Universidad Zaragoza

TRABAJO FIN DE GRADO

Procesado y exposición de datos sobre esports a través de un sistema de información

Autor:
Ramiro Woutersen Uriarte

Directores
María Piedad Garrido Picazo
Francisco José Martínez Domínguez

Escuela Universitaria Politécnica Teruel
2021

RESUMEN

Los esports cada vez son más competitivos. Existe una necesidad de obtener nueva información que permita contextualizar cada situación sobre su ámbito de permanente cambio, con tecnologías de Business Intelligence (BI).
Aquí es donde entra el Sistema de Información (SI) desarrollado en este Trabajo Fin de Grado (TFG) el cual obtiene datos sobre los resultados y los procesa para generar nueva información. Ésta es expuesta a través del sistema de gestión de contenidos Wagtail el cuál se ha desplegado en la plataforma como servicio Heroku. Todo este sistema tiene como lenguaje de programación base Python, el cual permite una integración más sencilla y con menos fallos de las diferentes herramientas y funcionalidades creadas.

PALABRAS CLAVE

Metajuego. Python. Wagtail. Heroku. Esports.

ABSTRACT

The esports industry is becoming increasingly competitive. There's a need to obtain new information that allows the contextualization of each situation on its area of permanent change, with Business Intelligence (BI) technologies.
Here is where the Information System (IS) developed in this Final Degree Project comes in, which obtains data from results and processes this data to generate new information. The new data is exposed through the content management system Wagtail which has been deployed on the platform as a service Heroku. This entire system has Python as its base programming language, which allows to get simpler integrations and fewer errors of the different tools and functionalities created.

KEYWORDS

Metagame. Python. Wagtail. Heroku. Esports.

Tabla de Contenidos

1.- Introducción

Este TFG consiste en el desarrollo de un Sistema de Información (SI) que permita recopilar, procesar y exponer datos sobre el entorno competitivo del deporte electrónico de "League of Legends". Este SI pretende definir una aplicación en cuatro capas donde su arquitectura está definida como: navegador, interfaz web, modelo y base de datos. Todo esto permite analizar los resultados de los partidos, a lo largo de los años, para poder ver de manera más clara una evolución de cómo se enfrentan los equipos a la competición y su desarrollo sobre la comprensión de juego.

1.1.- Motivación

Cuanto más se desarrolla un entorno competitivo más variables son tenidas en cuenta, y para comprender qué significan estos datos, es necesario que los conceptos que representan su significado estén bien definidos. Por ello, en entornos actuales de videojuegos uno de los conceptos que se definen es el concepto de metajuego.

Para entender este concepto primero hay que tener en cuenta cómo y porqué evoluciona el entorno de una competición. Esta evolución se debe no sólo al desarrollo de la concepción del propio juego, sino también por los posibles cambios de jugabilidad que pueden llegar a darse.

Por una parte está la concepción del juego, que provoca que las partidas se jueguen de manera diferente. Simplificando, un juego está formado por elementos y reglas. La comprensión sobre cómo pueden interactuar los elementos, entre ellos, a través de la reglas y como las diferentes acciones de los jugadores derivan en diferentes situaciones, provoca un entendimiento que permite determinar qué acciones son las más adecuadas para cada situación. A medida que avanza esta comprensión, avanza la manera de afrontar estas situaciones. Estas situaciones pueden tratar tanto temas estratégicos como temas de habilidad. Ejemplos de deportes no electrónicos serían el fútbol y el baloncesto, donde prácticamente toda la totalidad del avance sobre la comprensión del juego viene dado por el avance de su entendimiento sobre éste.

Por otra parte están los cambios sobre la jugabilidad, es decir, sobre los elementos o reglas del juego. Hay que tener en cuenta que los juegos usados para los deportes electrónicos tienen propietario. Si se califica la escena competitiva como un producto, el mantenimiento es una de sus partes clave en su ciclo de vida. Por ello y con el objetivo de alargar la vida del producto existen dos tipos de decisiones a la hora de actualizar estos juegos:

- **Cambios en el balance del juego:** Este tipo de cambios se debe a que los juegos pueden llegar a tener muchas reglas. Esto provoca que sea más difícil crear un sistema donde todas las opciones que se pueden escoger tengan el mismo impacto. Es decir, que puede que haya un elemento, una acción o una estrategia que llegue a ser más dominante que el resto. Con estos cambios el objetivo es acercar el juego a un estado de balance donde no exista una estrategia que pueda derrotar a la totalidad del resto de estrategias, es decir, una estrategia totalmente óptima o estrategia dominante.

- **Cambios en el desbalance del juego:** Aunque pueda sonar contra intuitivo, un juego perfectamente balanceado no es lo buscado en el diseño del juego. Los motivos son los siguientes:
    - Búsqueda de un juego para todos
    - Rotación de las estrategias dominantes

Esto cambios también suelen tener un enfoque sobre la audiencia de jugadores estándares (para la no competición). Que estos cambios se realicen también para la escena competitiva, se debe a que permite acercar el juego a la competición deportiva haciéndola más atractiva para el público de ambas partes. Por ejemplo, en el fútbol este efecto es más difícil de conseguir ya que para simular entre amigos un partido, con carácter competitivo, se requiere de la coordinación de veintidós personas más árbitros, creando una pequeña barrera de entrada para poder asemejar ambas experiencias.

Con este nuevo contexto, el metajuego se puede definir como un estado de cómo se entiende y juegan las partidas, **en un periodo de tiempo** relativo a la situación del juego. Estos metajuegos suelen definir las maneras óptimas en la que enfrentarse a los partidos, aunque hay que aclarar que estas maneras óptimas son deducidas de la comprensión del juego en el momento que se definen. Como se ha visto anteriormente, los deportes electrónicos tienden a actualizar sus reglas cada poco tiempo, dando que en una misma competición se utilicen versiones de juego diferentes, que se juegan de manera diferente. Esto provoca un metajuego no estancado y permite evitar que existan siempre las mismas estrategias dominantes.

Así pues la importancia sobre la capacidad de prever qué estrategias son las mejores en un momento determinado, gana mucha importancia. Incluso puede llegar a pasar que un equipo posea un estilo de juego fuerte y que, pasado unos meses de competición con varias actualizaciones, ese estilo de juego ya sea demasiado débil para afrontar el juego en el nuevo momento de la competición. Por ello y para tomar decisiones más informadas se requiere de un análisis de las evoluciones pasadas que permitan entender mejor como puede llegar a funcionar el juego.

1.2.- Objetivos

El objetivo principal es crear un sistema de Business Intelligence (BI) que permita producir y exponer datos de un contexto determinado.

Las capacidades del sistema propuesto que van a permitir la consecución del objetivo general son:

1. Realizar un proyecto de manera formal y estructurada.
2. Desarrollar un SI.
3. Encontrar proveedores de la información buscada.
4. Generar un sistema que obtenga los datos externos.
5. Utilizar alguna técnica de procesado que permita generar nueva información a partir de datos.
6. Crear un sitio web que permita exponer los datos.

Por otra parte, se encuentran los Objetivos de Desarrollo Sostenible (ODS), contribuyendo a la integración de la formación en competencias de Sostenibilidad en el Sistema Universitario Español.

Uno de los principales problemas sociales a nivel mundial es el desarrollo de la población en el ámbito intelectual. Por ello hay que promover una educación igualitaria la cual pueda permitir el avance socioeconómico de los diferentes países.

La innovación y el progreso tecnológico son claves para descubrir soluciones duraderas para los desafíos económicos y medioambientales, como el aumento de la eficiencia energética y de recursos. Esto permite generar empleo e ingresos los cuales tienen un papel clave a la hora de crear y promover nuevas tecnologías, facilitar el comercio internacional y permitir el uso eficiente de los recursos.

1.3.- Tecnologías escogidas

Los diferentes temas sobre las tecnologías escogidas y sus motivos son:

1. **Almacenamiento**

   Los posibles tipos de almacenamiento para el proyecto son el uso de bases de datos, tanto clásicas como NoSQL, o el uso de un cubo OLAP.

	BBDD SQL	BBDD No-SQL	Cubo OLAP
Enfoque en el almacenamiento	Todo tipo de volumenes de datos	Todo tipo de volumenes de datos	Grandes volumenes de datos
Tipos de trabajo	CRUD	Consulta para análisis	Consultas complejas para BI
Tipos de datos	Estructurados	Semiestructurados	Históricos
Complejidad del diseño	Media	Más baja	Más alta

Caracteristicas deseadas	
Cantidad de datos	Media
Tipos de trabajo	Consulta para análisis
Tipo de datos	Semiestructurados
	Históricos
Coste en tiempo	Bajo-Medio

El procesado se lleva a cabo en el modelo
Debido a la complegidad el cubo requiere mayor tiempo
No se pretende ni eliminar ni actualizar datos
Por ello se escoge una **BBDD No-SQL**

Figura 1 Comparativas entre tendencias de almacenamiento.

   Por lo tanto y a partir de los criterios expuestos se ha decidido usar un método de almacenamiento NoSQL.

2. **Lenguaje para el procesado**

   Para el procesado existen dos alternativas principales, usar Pyhton o R. En el caso de este TFG se ha decidido utilizar Python ya, que es el lenguaje más extendido. Esto conlleva que este más funcionalidades y más información al respecto.

3. **Desarrollo web**

Para la creación del sitio web del proyecto, se han considerado tres posibilidades: El uso de un framework, el uso de un CMS o el no uso de herramientas extra.

	Framework	Gestor de contenidos (CMS)	Base
Tiempo de aprendizaje	Alto	Media	Bajo
Complejidad esperada	Media	Baja	Media
Modelo de trabajo	Desarrollo sobre funcionalidad	Desarrollo sobre contenido	Desarrollo arbitrario

Caracteristicas deseadas	
Tiempo de aprendizaje	Medio
Tipo de trabajo	Creación de contenido
Coste en tiempo	Bajo-Medio
Compatibilidad	Mayor posible

Se busca el menor tiempo de desarrollo posible
Mayor compatibilidad posible entre web y modelo
Debido a que la compatibilidad y la complejidad esperada pueden llegar a depender de la herramienta se decide explorar diferentes opciones para los todos casos

Figura 2 Software de base.

1.4.- Herramientas

Las herramientas específicas a utilizar son:

Gestor de DB	Bibliotecas especializadas	Entorno de desarrollo	Cms/Framework	Web
Robot3T	Numpy	Notepad++	Wagtail	Django
Atlas	Pandas	Visual Studio Code		Plantilla
	Mwclient			
	Bokeh			

Figura 3 Temas y herramientas.

Para el uso de un gestor de BBDD se ha utilizado, Robot3T de manera local y Atlas de manera remota. Las bibliotecas Numpy, Pandas y Bokeh están especializadas en el tratamiento de datos mientras que por otra parte mwclient se especializa en la conexión remota a una fuente de datos.
Como entorno de desarrollo para la generación de código para el recolector y el procesador se ha utilizado Notepad++, mientras que para el CMS se ha trabajado con una herramienta más potente, Visual Studio Code. En el contexto web se ha utilizado una plantilla bootstrap para generar una base y Django para desarrollar funcionalidades más específicas.

La herramienta más relevante ha sido Wagtail, la cual es un CMS más framework con la cual se ha generado el entorno web.

2.- Estado del Arte

Durante la última década los deportes electrónicos han ganado relevancia. Las bases de esta industria se han visto reforzadas en sus tres pilares básicos: financiación, visibilidad e infraestructura.

Por una parte, los inversores ven esta industria como una gran oportunidad y aunque, se ha demostrado que este tipo de inversiones son de alto riesgo, debido a la naturaleza primeriza de esta industria, la tendencia al alza de gasto es similar a la de los grandes deportes tradicionales. Por otra parte, está la visibilidad que aporta este producto. El público objetivo está bastante definido (joven) y permite la posibilidad subyacente de introducir tanto marcas como patrocinios. Por último, está la infraestructura tanto física como de recursos humanos que se ha ido generando lentamente y que, a día de hoy, en algunos deportes electrónicos ya tiene un impacto y desarrollo elevado, similar a deportes tradicionales.

Este último punto es relevante para el tema tratado en este TFG. Tanto el desarrollo de nuevos jugadores, como el conocimiento de los cuerpos técnicos de los diferentes equipos se nutren del conocimiento generado durante estos años, y del entendimiento potencial del juego a futuro.

Dentro de los esports existen una gran variedad de juegos. A estos se les puede categorizar por varios puntos:

- **Género:** El género en un juego es definido principalmente por cómo se juega, a diferencia de otros campos artísticos donde el género lo indica puramente la experiencia evocada por la obra. Es decir que suele representar aspectos técnicos de la jugabilidad.
- **Envergadura:** Este punto hace referencia al tamaño del deporte en términos productivos. Como ejemplo, la petanca es un deporte con poco público mientras que el fútbol sería un deporte con mucho público.
- **Potencial deportivo estimado:** Está claro que el conjunto de deportes tradicionales hoy en día tiene un carácter estable. Los juegos en los que se compite son los mismos desde hace ya décadas. Esto es diferente en los esports donde existen diferentes juegos con géneros muy distintos. El potencial hace referencia principalmente a dos puntos. Por una parte el potencial como entretenimiento. Aquí entrarían factores como, que un espectador entienda lo que pasa en el juego con facilidad o el interés que se tiene por éste. Por otra parte está el potencial competitivo donde el juego tiene que ser capaz de generar un entorno de competición adecuado. Es decir, que se consigan generar un conjunto de escenarios con resoluciones polivalentes lo suficientemente interesantes.

Algunos de los esports que existen en la actualidad son:

	Género	Envergadura	Potencial deportivo
Hearthstone	Juego de cartas	Media	Medio
Smash Brosh Ultimate	Lucha	Baja	Medio
CS: GO	Disparos	Alta	Medio-Alta
Fortnite	Disparos	Alta	Medio
Overwatch	Disparos	Media	Medio-Bajo
League of Legends	Estrategia	Alta	Alto
Dota 2	Estrategia	Alta	Alto

Figura 4 Estado de los esports [1].

A día de hoy, dentro de esta industria, hay diferentes deportes electrónicos donde cada cual tiene sus fortalezas y sus debilidades. Como se pude intuir por el tema del proyecto, una de las fortalezas principales de la competición elegida es su infraestructura. Caracterizado tanto por su talento existente o a generar como por su bagaje en el desarrollo del entendimiento del juego. El juego tratado es el "League of Legends", el que cuenta con una escena competitiva que existe desde hace ocho años, contribuyendo tanto al desarrollo conceptual como económico del juego. Pero para entender las diferentes ideas y conceptos lo mejor es empezar desde el principio.

¿Qué es y cómo funciona el Legue of Legends?
El League of Legends es un juego para ordenador. Éste es jugado por dos equipos, de cinco personas, donde cada equipo controla una zona aliada y su objetivo es destruir la zona enemiga. El juego contiene lo que se llaman *campeones*, que son los personajes usados por cada jugador. La cantidad de campeones que se incluyen en el juego son más de cien, y antes de iniciarse una partida cada jugador elige con que campeón va a jugar.

Para precisar más en esta fase anterior a una partida, se utiliza un sistema de 'draft' con 'banns'. Es decir, los jugadores se turnan para elegir personaje y para restringir campeones, haciendo que estos últimos no puedan ser escogidos durante esta partida.

Figura 5 Fase de elección de campeones pre-partido.

El lugar virtual donde se desarrolla el juego es conocido como "mapa". Este mapa es cuadrado y en él se encuentran las zonas de cada equipo. Estas zonas son denominadas como "bases" y están unidas por tres caminos, dejando unas zonas intermedias entre estos caminos llamados "jungla".

Al igual que en muchos deportes tradicionales, cada jugador tiene asignados una posición y un rol. Pero en este caso la relación entre posición y rol no están estrictamente relacionados. Existen cinco posiciones y son consideradas en inglés (incluido en otros países, e introducidas como anglicismos). Estas son:

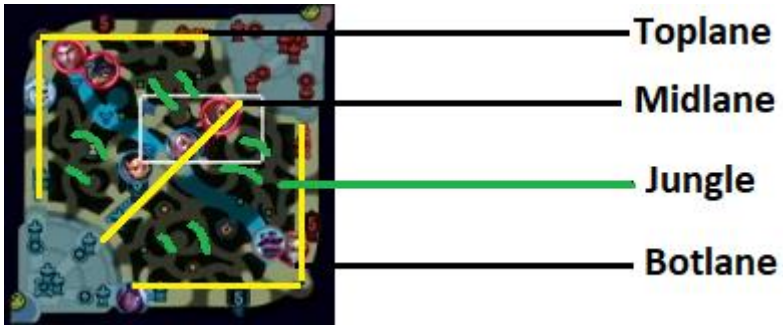

Figura 6 Mapa de League of Legends.

Es decir, cada jugador antes de la partida escoge un campeón con un rol acorde al planteamiento dando que cada partida se juegue, normalmente, con campeones diferentes.

También existen otros elementos del juego como oro, objetivos y objetos, que también influyen en el planteamiento de cada partida.

Para resumir, cada dos semanas las reglas del juego cambian modificando valores tanto de campeones como de otros elementos existentes. Estos cambios provocan que la prioridad sobre el tipo de planteamiento de partida y sobre qué campeones están asociados a este planteamiento fluctúe de manera constante. Dando información suficiente y creando una necesidad para un análisis de la situación del juego, o más concretamente, del estado del juego.

3.- Planificación del proyecto

3.1.- Ámbito

La situación personal se ha caracterizado y visto afectada por dos puntos.

1. **La pandemia:** El Covid-19 y sus restricciones de movilidad tienen implicaciones en temas de rutinas y acciones de continuidad. La pandemia puede afectar de manera psicológica al desarrollo del proyecto.
2. **Fin de la Carrera:** Después de varios años cursando asignaturas impartidas de manera directa, el final de la carrera se caracteriza por un proyecto final. A diferencia del resto de los cursos académicos, este TFG (como asignatura) se caracteriza por la autoguía que hace el alumno a través de la mayor parte del trabajo. La idea principal es utilizar los conocimientos adquiridos entre el que se incluye el conocimiento para adquirir más conocimiento. Teniendo también en cuenta la generación de SI, la organización y estructuración de proyectos. De manera más concreta ya se ha trabajado con bases de datos no relacionales, con tecnologías web y con el lenguaje Python, aunque de manera reducida y académica, con una aplicación concreta. De esta manera parte del desafío es poner en práctica de manera conjunta y ampliar los conocimientos sobre los temas a utilizar.

3.2.- Viabilidad

La viabilidad del proyecto representa la posibilidad estimada de que éste se consiga llevar a cabo. Para ello se necesitan metas realistas y adecuadas. Por lo tanto se van a considerar tres puntos principales que definen las limitaciones del trabajo.

- **Tiempo:** El tiempo máximo para la realización de este TFG sería el equivalente al del transcurso de un curso lectivo, es decir, dos cuatrimestres. En este caso en particular equivaldría a un mes menos debido a que la fecha de inicio de trabajo fue en Octubre.
- **Dificultad:** La dificultad tanto conceptual como técnica del proyecto ha de ser acorde a los conocimientos adquiridos durante la carrera. Por una parte, se han de poner en práctica los conocimientos generales de la carrera (estructuras de programación, resolución ingenieril de problemas, estructuración de proyectos) y por otra parte, los conocimientos de la rama específica cursada (SI, desarrollo de proyectos formales, tecnologías web).
- **Envergadura del proyecto:** La envergadura del proyecto hace referencia a la cantidad de trabajo esperado. Por lo tanto, hay que tener en cuenta que el uso de nuevas tecnologías contribuye más a esta envergadura frente al uso de tecnologías ya familiarizadas.

Como se puede intuir, que las diferentes tecnologías utilicen el mismo lenguaje de base, Python, permite reducir tanto el tiempo como la dificultad del proyecto.

3.3.- Análisis de Riesgos

Los principales riesgos del proyecto son:

- **Problemas de conexión entre tecnologías y módulos:** Aun escogiendo herramientas que utilizan el mismo lenguaje de programación, la inclusión de sus funcionalidades al total del sistema o entre éstas, puede ser un problema que bloquee la realización del proyecto. Dentro de las diferentes posibilidades los puntos con más riesgo han sido la exposición de las gráficas Bokeh a través del CMS y el uso de funcionalidades externas en el CMS. Es decir, la conexión del CMS con herramientas externas. Por otra parte, el

uso de la base de datos es el punto con menos riesgo, principalmente porque ya se tiene una experiencia mínima para la gestión de ésta.

- **Posible falta de documentación:** Uno de los problemas es la falta de documentación de las herramientas a utilizar. En principio para cuestiones más generales no debería de haber ningún problema ya que Python, Mongo, Pandas y Bokeh tienen documentaciones muy amplias. Pero por otra parte el uso de Wagtail, como CMS, y aunque tenga una comunidad de usuarios suficientemente amplia no garantiza que exista documentación para algún problema determinado.

- **Dificultad añadida:** No sólo una posible falta de documentación puede ser motivo de las limitaciones a la hora de implementar. También influye el conocimiento aún no adquirido y cómo afecta a la hora de buscar la información necesaria. Es decir, qué intentar buscar soluciones a problemas, en un contexto determinado, puede provocar que se busque una cuestión de una forma pero que las palabras utilizadas o la composición de la frase en cuestión no sea la que lleve a la solución. Pero sí que exista una pregunta similar con otros tecnicismos concretos.

3.4.- Planificación temporal

Para la planificación temporal se ha utilizado la herramienta de GnattProject [2]. Primero se ha establecido una previsión inicial.

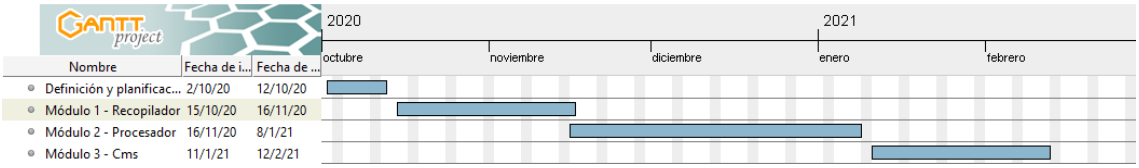

Figura 7 Previsión inicial sobre la planificación temporal.

Pero claro, como suele darse en este tipo de proyectos el tiempo para el desarrollo de éste ha acabado siendo mayor. Aproximadamente un 50% mayor. Esto se da por la necesidad de trabajar en varios puntos bastante diferentes y no totalmente conocidos.

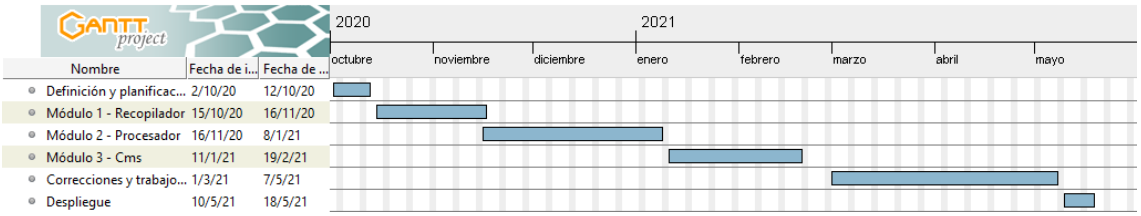

Figura 8 Resultado final de la planificación temporal.

Como se puede ver se ha forzado el cumplir el tiempo asignado inicialmente por lo que el punto quinto de correcciones y trabajo mixto ha sido principalmente la realización de funcionalidades necesarias, no finalizadas en su tiempo asignado, y la corrección de errores .

4.- Análisis del proyecto

4.1.- Requisitos (funcionales y no funcionales)

Los requisitos propuestos siguen la norma ISO/IEEE 830 [3].

- **Requisitos Funcionales**

  Base de datos:

  RQF 01 El sistema almacena datos recopilados y datos procesados.
  RQF 02 La base de datos puede ser accesible de forma remota con facilidad.

  Eficiencia:

  RQF 11 El sistema es capaz de recopilar datos obtenidos de un proveedor externo al sistema.
  RQF 12 El sistema es capaz de generar nuevos datos a partir de datos almacenados ya existentes.
  RQF 13 El sistema es lo suficientemente autónomo como para seguir funcionando incluso si fallan los proveedores de datos.

  Interfaces de usuario:

  RQF 21 Existe la capacidad de que un usuario avanzado, sin altos conocimientos de informática, pueda crear contenido asociado a la web.
  RQF 22 El sistema genera contenido que está expuesto de manera ordenada.
  RQF 23 Es posible que la exposición de los datos se suministre en forma visual, es decir, con gráficos y/o tablas.
  RQF 24 El sistema es modular, de manera que a la hora de cambiar una de las tecnologías usadas, no afecte al resto de partes de la interfaz.

- **Requisitos No Funcionales**

  Eficiencia:

  RQNF 01 El sistema debe de ser capaz de estar activo 24h los siete días de la semana, quitando en situaciones excepcionales como mantenimientos o caídas no deseadas.
  RQNF 02 El sistema debe de ser capaz de albergar la totalidad de la información, a obtener de los torneos en una misma base de datos.
  RQNF 03 El tiempo de procesamiento de una función no debe de ser mayor a 24 horas.
  RQNF 04 Los tiempos de carga para las interfaces de usuario son de menos de 5 segundos.

  Accesibilidad:

  RQNF 11 Las interfaces de usuario son perceptibles, operables y comprensibles.
  RQNF 12 El conjunto de los colores y los tamaños de fuentes de las interfaces son visibles y distinguibles para los usuarios.

  Usabilidad:

  RQNF 21 La legibilidad de la interfaz permite distinguir de manera clara y concisa las diferentes partes.
  RQNF 22 Existe una gama de colores de la interfaz que permite distinguir de manera clara y concisa las diferentes partes.
  RQNF 23 El estado de la interfaz es en todo momento visible y se sabe en qué zona se encuentra uno en todo momento.

5.- Diseño del proyecto

Uno de los puntos principales del proyecto es la creación de un SI sencillo, pero autosuficiente. Debido a la varianza de la envergadura del proyecto se proponen dos estructuras, siendo una de ellas una versión mejorada de la anterior.

Como todo SI existen tres partes fundamentales: la base de datos, un recopilador de datos, y un trabajador de datos. En este caso al trabajador de datos se ha considerado como procesador de datos. Por otra parte, está la exposición de los datos de la que se encarga la parte web. Esta parte está formada por una parte "back-end" y otra "front-end".

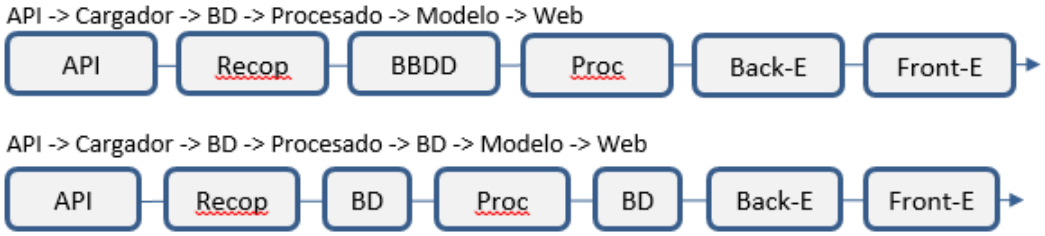

Figura 9 Secuencia de diseño inicial y mejorada.

Se han generado unos diagramas simples que representan la estructura de cada módulo. El módulo primero es sobre la recolección de datos, el módulo segundo sobre el procesado y el módulo tercero sobre la exposición.

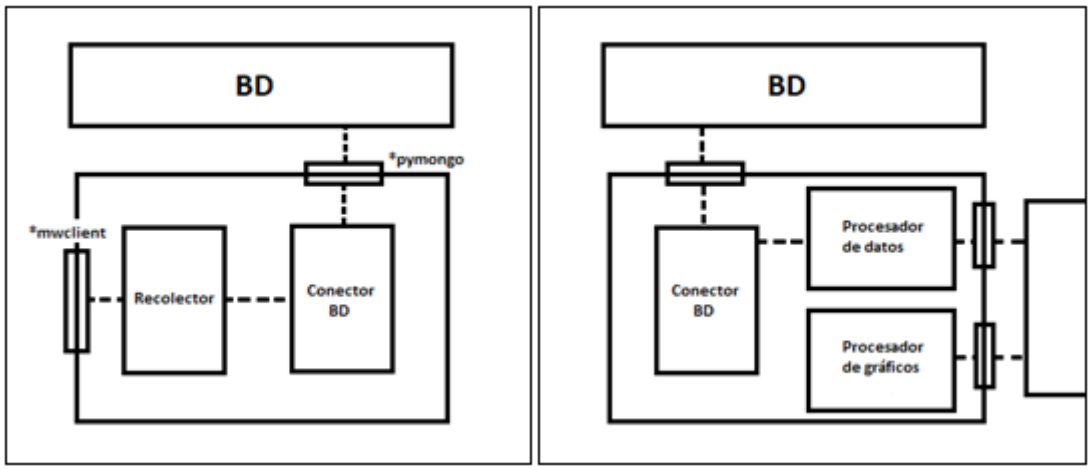

Figura 10 Módulo 1 (Recopilador) y 2 (Procesador) ordenados de izquierda a derecha respectivamente.

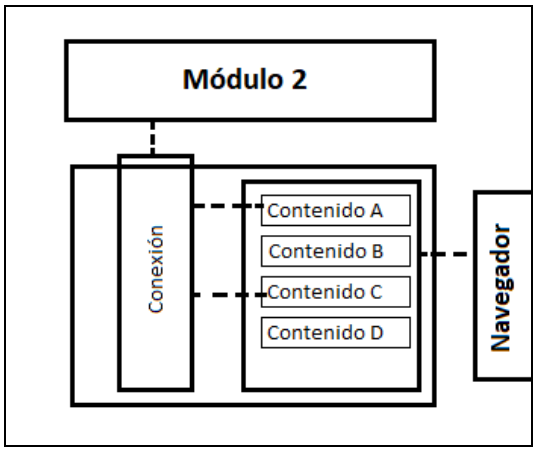

Figura 11 Módulo 3 (Expositor).

6.- Desarrollo del proyecto

El desarrollo del proyecto se ha llevado a cabo primero por bloques, y después en un trabajo conjunto sobre los diferentes bloques.

6.1.- Base de datos local

Entorno de trabajo:

Para el uso local de una base de datos NoSQL se ha seleccionado como tecnología de base de datos Mongodb, y como tecnología de Gestor de Base de Datos Robo 3T.

- Instalación Mongodb:
  La instalación de Mongodb se puede conseguir a través de un ejecutable. Por lo tanto, los pasos son la descarga del ejecutable y la ejecución de éste.
- Instalación Robot 3T:
  La instalación de Robot 3T se puede conseguir a través de un ejecutable. Por lo tanto los pasos son la descarga del ejecutable y la ejecución de éste.

Una vez instalada la base de datos hay que asegurarse de iniciar el servicio de Mongo.

- Ejecutar desde la ruta adecuada o incorporar mongod a las variables de entorno del sistema.
- Ejecutar con permisos de administrador, --dbpatch apunta a la base de datos.

```
C:\Program Files\MongoDB\Server\4.2\bin>mongod --port 27017 --dbpath "C:\Program Files\MongoDB\Server\4.2\data"
```

Figura 12 Iniciar el servicio de MongoDB.

6.2.- Recopilador de datos

El primer punto a tratar sería la obtención de los datos base sobre los diferentes resultados deseados. Para ello se han explorado y encontrado cuatro fuentes de información:

- **Resultados por página web**
  La empresa propietaria publica en forma de página web el resultado de cada partido único. Estos resultados se encuentran con llamadas en forma de http y son los usados para consultar de manera directa un partido. Pero para utilizar este sistema existen varios problemas:
  a) A cada partido le corresponde un código id numérico asociado no representativo de su url, por lo que habría que recopilar de alguna otra parte estos ids.
  b) Por otra parte el problema sería que habría que leer de manera personalizada de la página los resultados, como leyendo texto de una página web.

  Esto provoca que haya que implementar dos sistemas en vez de uno, para la obtención de ids y para la recepción de la información.

- **Resultados de la api**
  Riot Games también proporciona un api para ser usada por los desarrolladores. Pero por lo visto hace ya unos años que los partidos de ligas profesionales han sido recolocados en un servidor privado. Por lo que sólo sirve para obtener partidos amateurs.

- **Oracle's Elixir**
  Esta fuente es la página web gestionada por una persona. Aquí se pueden encontrar los resultados de los últimos años en formato csv. El principal problema es la

dependencia de la fuente, que al ser un particular es bastante elevada. También existe un inconveniente, el formato. Este formato de datos está preparado y listo como información procesada, por lo que habría que filtrar estos datos antes de colocarlos en la base de datos del TFG.

- **Lolpedia, ahora renombrado a Leaguepedia**
  Lolpedia es un sitio web con formato de 'wiki', lo que significa que las páginas son editables por diferentes grupos de personas. También es una página de uso habitual para la consulta de datos de la escena, y no sólo de resultados de partidos (fichajes, movimientos, historia de equipos, jugadores, etc...).
  Esto permite evitar un riesgo en la dependencia, es decir, es menos probable que deje de funcionar y/o de que contenga datos erróneos. Los datos de esta página se pueden obtener mediante consultas. El principal inconveniente es la necesidad de transformar estos datos de estructura SQL (Structured Query Language) a JSON (JavaScript Object Notation).

Dentro de las diferentes fuentes encontradas, la más adecuada es la de Lolpedia. Necesita un trabajo extra de transformación de estructura de datos pero los demás puntos son ventajas. Se puede automatizar el proceso, evitando no necesitar tocar nada a mano, y los datos son completos, a diferencia de Oracle's Elixir que no contiene datos más antiguos.

Para realizar consultas a los datos de Lolpedia existen varias maneras, encontrando una librería en Python, que facilita este proceso.

Esta librería se llama “mwclient” y permite realizar consultas a los sitios web que poseen una estructura del tipo 'gamepedia' (como Lolpedia). Esto también permite tener nueva información potencial de otros deportes, para el futuro. El funcionamiento de este paquete es sencillo, a partir de una URL se crea una variable tipo objeto que permite hacer las consultas. Lo complicado viene a la hora de obtener los datos adecuados, transformarlos al formato deseado y colocarlos de forma correcta.

Lo primero, es acostumbrarse y ponerse en contexto de la estructura de datos que tiene Lolpedia. Las tablas principales en las cuales se encuentran los datos deseados son:

- Tournaments
- TournamentRosters
- ScoreboardGames
- ScoreboardPlayers
- MatchSchedule

El siguiente paso es la definición de la estructura de los datos recolectados para la base de datos propia. Para ello se han tenido en cuenta tres opciones y sus eficiencias en dos operaciones distintas:

```
---Opción 1: Plana (similar a la fuente)---
{
        "tournament": "LEC 2020 Summer Playoffs",
        "matches": [
                {
                        "team1": "G2",
                        "team2": "FNC",
                        "team1picks": ['Aatrox', 'Lillia', 'Akali', 'Draven', 'Nautilus'],
                        "team2picks": ['Shen', 'Hecarim', 'Syndra', 'Senna', 'Thresh'],
                        "team1players": ['Wunder', 'Jankos']
                        "team1playerskills": [3,5,2,4,0],
                        "team2playerskills": [1,0,5,2,3]
                }
        ]
}
=> Obtener composiciones
tournaments = db.tournaments
for tournament in tournaments.find({},{'tournament': 'LEC 2020 Summer Playoffs', '_id': 0}):
  for match in tournament['matches']:
        print(match['team1picks'])

=> Obtener kills de un equipo
for tournament in tournaments.find({},{'tournament': 'LEC 2020 Summer Playoffs', '_id': 0}):
  for match in tournament['matches']:
        print(match['team1playerskills'])
```

Figura 13 Primera opción de estructura de datos interna.

```
---Opción 2: La información inyectada de ScoreboardPlayers indexada---
{
        "tournament": "LEC 2020 Summer Playoffs",
        "matches": [
                {
                        "team1": "G2",
                        "team2": "FNC",
                        "team1picks": ['Aatrox', 'Lillia', 'Akali', 'Draven', 'Nautilus'],
                        "team2picks": ['Shen', 'Hecarim', 'Syndra', 'Senna', 'Thresh'],
                        "team1players": [
                                {
                                        "name": "Wunder",
                                        "kills": 3

                                },
                                {
                                        "name": "Jankos"
                                        "kills": 5
                                }
                }
        ]
}
=> Obtener composiciones
tournaments = db.tournaments
for tournament in tournaments.find({},{'tournament': 'LEC 2020 Summer Playoffs', '_id': 0}):
  for match in tournament['matches']:
        print(match['team1picks'])

=> Obtener kills de un equipo
for tournament in tournaments.find({},{'tournament': 'LEC 2020 Summer Playoffs', '_id': 0}):
  for match in tournament['matches']:
        for player in match['team1players']:
                print(player['kills'])
```

Figura 14 Segunda opción de estructura de datos interna.

```
---Opción 3: Toda la información posible indexada---
{
        "tournament": "LEC 2020 Summer Playoffs",
        "matches": [
                {
                        "team1":
                        {
                                "name": "G2",
                                "players": [
                                        {
                                        "name": "Wunder",
                                        "champion": "Aatrox",
                                        "kills": 3

                                        },
                                        {
                                                "name": "Jankos"
                                                "champion": "Lillia",
                                                "kills": 5
                                        }
                                ]
                        },
                        "team2":
                        {
                                ...
                        }

                }
        ]
}
```

```
=> Obtener composiciones
tournaments = db.tournaments
for tournament in tournaments.find({},{'tournament': 'LEC 2020 Summer Playoffs', '_id': 0}):
  for match in tournament['matches']:
        for player in match['team1'][0]['players']:
                print(player['champion'])

=> Obtener kills de un equipo
for tournament in tournaments.find({},{'tournament': 'LEC 2020 Summer Playoffs', '_id': 0}):
  for match in tournament['matches']:
        for player in match['team1'][0]['players']:
                print(player['kills'])
```

Figura 15 Tercera opción de estructura de datos interna.

Como se puede observar, las estructuras más anidadas hacen que se requiera de más recorrido (sobre el conjunto de datos) para encontrar la misma información. Por ello la opción escogida para la estructura de la base de datos, contenedora de la información sin procesar, es la "Opción 1".

Una vez está definida la estructura deseada, se pasa a trabajar con la recolección de los datos. La idea principal es que primero se recolectan los datos de las diferentes tablas, después se restructuran y por último se insertan.

Los principales desafíos encontrados a lo hora de la implementación son los siguientes:

- **Estructura de datos que se obtiene**
  El tipo de estructura de datos que se obtiene de Lolpedia tiene tipo de 'OrderedDict'. Por lo que hay que encontrar la forma adecuada para transformar los datos y poder usarlos.

```
[OrderedDict([('title',
               OrderedDict([('Team', 'Excel Esports'),
                            ('RosterLinks',
                             'Expect;;Caedrel;;Mickey (Son '
                             'Young-min);;Patrik;;Tore;;YoungBuck;;Mapache'),
                            ('Region', '')]))]),
 OrderedDict([('title',
               OrderedDict([('Team', 'FC Schalke 04 Esports'),
                            ('RosterLinks',
                             'Odoamne;;Gilius;;Lurox;;Abbedagge;;FORG1VEN;;Innaxe;;Dreams '
                             '(Han Min-kook);;Dylan Falco'),
                            ('Region', '')]))]),
 OrderedDict([('title',
               OrderedDict([('Team', 'Fnatic'),
                            ('RosterLinks',
                             'Bwipo;;Selfmade;;Nemesis;;Rekkles;;Hylissang;;Mithy;;Aagie'),
                            ('Region', '')]))]),
```

Figura 16 Resultado OrderDict de obtener los equipos, las. Plantillas y la región de un torneo (LEC 2020 Spring).

OrderDict es un tipo de datos usado en Python que se puede recorrer, por lo que es sencillo generar la nueva estructura deseada. Con el ejemplo anterior se añaden los equipos a los datos de un torneo.

```python
#Add teams data to tournament (not to match)
for row in query_teams:
    team = {}
    team['name'] = row['title']['Team']
    team['roster'] = row['title']['RosterLinks'].split(';;')
    if query_region[0]['title']['Region'] == 'International':
        team['region'] = row['title']['Region']
    data['teams'].append(team)
```

Figura 17 Código desarrollado para la adición de la información de los equipos a los datos de un nuevo torneo.

- **Limitación del tamaño de query**
  Una de las limitaciones impuestas por la base de datos de Lolpedia, es la cantidad de datos que puede contener una query. Está limitado a 500, por lo que se implementa una solución de iteración acorde. En esta solución se hace una query extra para contar cuantos datos hay, por lo que el estrés que se provoca en el proveedor es ligeramente mayor (aunque obtener cuantos datos hay no tiene mucho peso computacional, por tiempo).

  Limits [edit]

  - 5,000 results per query if you are an admin
  - 500 result per query otherwise

  Figura 18 Limite Query Leaguepedia [4]

- **Último partido de una serie**
  No existe forma directa para saber si un partido en una serie es el último que se ha jugado. Por ejemplo, en una serie al mejor de cinco partidos, gana el primer equipo que salga victorioso de tres partidos. De esta manera los resultados pueden ser 3-0, 3-1 o 3-2.

Para solucionar este problema se decide calcular 'al vuelo' si es el último partido jugado de la serie. Se hace comprobando que cualquiera de los equipos haya llegado a la mitad entera más uno de partidos.

Una vez completado el recolector de datos y superados los principales problemas se pueden ver los datos dentro de la base de datos:

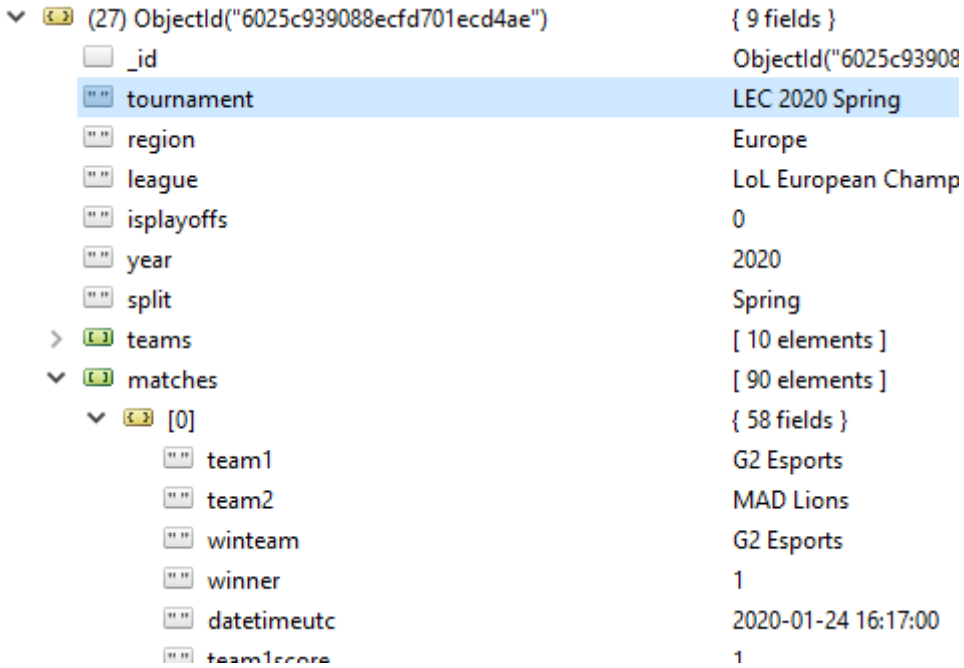

Figura 19 Muestra de datos de un partido de un torneo a través de Mongo3T.

6.3.- Procesador de datos

Para empezar a crear el procesador de datos lo primero es instalar las herramientas necesarias. En este caso son Numpy, Pandas y Bokeh. En entornos Python, comúnmente, se utiliza una herramienta llamada Pip. Esta es un gestor de paquetes Python. La instalación se puede hacer desde la consola de comandos.

Descarga; *"curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py".*
Instalación; *"python get-pip.py".*

Una vez instalado Pip se puede introducir por la consola de comandos "pip install nombre" lo que facilita la instalación de los paquetes. En este TFG se introduce para Numpy, Pandas y Bokeh.

Con la instalación del entorno de trabajo completa se ha dividido la tarea a realizar en tres partes:

Qué datos buscamos:
-datos de análisis y predicción
El primer punto es saber qué información queremos generar. En este caso se ha decidido trabajar en dos ámbitos. Datos de análisis y datos de predicción. Dentro del tema del TFG los puntos que permiten generar una base de información sólida son:

- **Equipos:** La información de los resultados de los equipos permite obtener una fotografía de uno o varios puntos en el tiempo sobre como de acertados son/fueron

sus enfoques estratégicos. Esto se puede ver en diferentes factores como duración de partida, sobre que prioridades se están enfocando o el rendimiento de jugadores. También se puede observar los resultados de enfrentar diferentes estrategias.

- **Campeones/roles:** Este conjunto de puntos permite ver, de manera más concreta, la estrategia composicional del metajuego dando una dirección estimada sobre qué rumbo está tomando.

Procesado de datos:
Para la conexión a la base de datos se implementa de manera similar al recolector, gracias a que se comparte el mismo lenguaje de programación.
Por temas de distribuir y modular los datos se ha decidido separar los datos "base" obtenidos con los datos procesados. Así que hay que crear tanto una conexión de entrada como de salida.

La fase de implementar las funcionalidades del procesado se ha caracterizado por un desarrollo incremental por parámetros. Es decir, a partir de una función que procesaba algo sencillo, se ha ido modificando esa función para que generase nuevos datos más precisos a partir de una mayor cantidad de parámetros.

Las funciones finales permiten generar: los resultados de los equipos por distintos tipos de agrupaciones, los ratios de victoria de los campeones y la cantidad proporcional de los roles por posición.

	team	dragons	enemydragons	dragonsrate	games
8	SK Gaming	58	48	3.22222	18
7	Rogue (European Team)	54	45	3	18
4	MAD Lions	51	35	2.83333	18
9	Team Vitality	45	40	2.5	18
0	Excel Esports	44	51	2.44444	18
2	Fnatic	43	50	2.38889	18
3	G2 Esports	42	62	2.33333	18
1	FC Schalke 04 Esports	40	48	2.22222	18
5	Misfits Gaming	39	40	2.16667	18
6	Origen	37	34	2.05556	18

Figura 20 Estadísticas sobre dragones.

Generación de gráficas:
Para la generación de gráficas en Bokeh hay que tener en cuenta la figura. Ésta es la variable que engloba el conjunto de la gráfica y a partir de ella se añaden características para generar la configuración deseada.

Aunque con este tipo de gráficas ya se ha trabajado en alguna asignatura del Grado con anterioridad, esto ha sido de manera más sencilla y directa. Para la implementación de gráficas algo más complejas también se han buscado ejemplos de los cuales se ha aprendido de manera más detallada su funcionamiento. Destacar que se ha consultado mayoritariamente la documentación oficial, a diferencia de los problemas tratados, como por ejemplo para obtener información sobre el uso de colores [5]. Dentro de las gráficas que se ha conseguido crear de manera modular se encuentran: de barras, de puntos y de áreas.

```
from bokeh.palettes import brewer, cividis, all_palettes, viridis, plasma
...
colors = plasma(df[df.columns.values[1]].size)

index_cmap = factor_cmap('name', palette=colors,
                         factors=sorted(df.name.unique()))
```

Figura 21 Creación y asignación de una paleta de colores a los valores de una gráfica Bokeh.

6.4.- CMS (Content Management System)

Para instalar Wagtail lo primero es comprobar que se tiene Python y Pip instalado. Después simplemente se hace la instación del cms por Pip con el comando "pip install wagtail".

```
C:\Users\Usuario\Downloads\Activo>python --version
Python 3.8.6

C:\Users\Usuario\Downloads\Activo>pip --version
pip 20.2.1 from c:\users\usuario\appdata\local\prog
```

Figura 22 Comprobación de las versiones de Python y Pip, indicando que están instaladas.

Una vez instalado, para crear un nuevo proyecto se introduce "wagtail start mysite mysite". También se recomienda navegar dentro del directorio del proyecto y crear un archivo de requerimientos. "pip install -r requirements.txt"

Para realizar la configuración inicial hay que realizar tres ajustes por línea de comandos:

```
python manage.py migrate
python manage.py createsuperuser
python manage.py runserver
```

Pero, ¿Qué es lo que hacen estos comandos? Primero hay que saber que el CMS guarda dos tipos de datos: estructurales y concretos.

1. Estructurales, hacen referencia a la estructura del sistema.
2. Concretos, hacen referencia al contenido de la estructura del sistema.

Las herramientas de framework/CMS contienen una base de datos que guarda la información sobre estos. Por lo que los datos estructurales hacen referencia a la definición de cada tabla y campo. Por otra parte los concretos hacen referencia al contenido insertado a los campos que se generan en estas tablas.

	page_ptr_id	banner_title	banner_cta_id	banner_image_id	banner_subtitle	content
	Filter	Filter	Filter	Filter	Filter	Filter
1	3	Welcome to Sta...	NULL	2	<p>We help sta...	[{"type": "cta",...

Figura 23 Representación de una página en la base de datos [6].

Los comandos para gestionar el CMS son los siguientes:

a) ***python manage.py makemigrations***:
   Si se han realizado cambios estructurales del CMS hay que indicar estos cambios.

b) ***python manage.py migrate***:
   Si hay cambios indicados sobre la estructura de algún elemento del CMS se generan estos cambios dentro de la bd.

c) ***python manage.py runserver:***
Se pone en marcha el CMS (con la estructura de la base de datos, definida por la última migración del sistema).

d) ***python manage.py createsuperuser***:
A partir de un nombre de usuario, correo electrónico y contraseña se crea un usuario con privilegios.

Una vez instalado Wagtail, el primer punto es aprender los conocimientos básicos sobre éste. Tanto su funcionamiento estructural, como sus utilidades.

Funcionamiento estructural:
El primer punto para entender el modelo de datos es diferenciar entre las dos referencias conceptuales a página que utiliza el CMS. Por una parte está la definición de la página, donde se guardan las características generales de las mismas. Por otra parte, se encuentra la construcción real de estas páginas, las cuales derivan su información de estas definiciones y se agrupan en forma de árbol. Lo que vendría a ser una clase y su instanciación.

```python
class ShowerPage(Page):
    """Flexible page class."""

    template = "shower/shower_page.html"
    subpage_types = ['shower.ShowerPage']
    parent_page_types = [
        'shower.ShowerPage',
        'home.HomePage',
    ]
    content = StreamField(
        [
            ("title_and_text", blocks.TitleAndTextBlock()),
            ("full_richtext", blocks.RichtextBlock()),
            ("simple_richtext", blocks.SimpleRichtextBlock()),
            ("char_block", streamfield_blocks.CharBlock(
                required=True,
                help_text='Oh wow this is help text!!',
                min_length=10,
```

Figura 24 Definición de una página.

SORT	TITLE	UPDATED	TYPE	STATUS
	Equipos	0 minutes ago	Shower Page	LIVE
	Regiones	1 week, 5 days ago	Shower Page	LIVE

Figura 25 Dos páginas del mismo tipo creadas en el CMS.

Es decir, con un tipo definido, se pueden crear múltiples páginas de ese tipo. Las principales características con las cuales se van a trabajar son:

- “content_panels”: Tipos de los contenidos de la página.
- “template”: Html específico asociado.

Streamfield y bloques:
Una página de Wagtail está contenida por 'fields'. Estos están heredados de Django y permiten definir un tipo de contenido que se asigna a la definición de una página. La principal diferencia de Wagtail con sus competidores es el tipo de dato 'Streamfield'. Este tipo de dato es un campo 'field', similar al usado en Django, y se caracteriza por contener a su vez unos tipos de datos llamados **bloques**. Este elemento permite componer un 'field' contenido por un conjunto de bloques determinado, tanto en tipo como en orden.

La ventaja de 'Streamfield' es que provee al creador de contenido de una mayor libertad y una mayor cantidad de herramientas para usar.

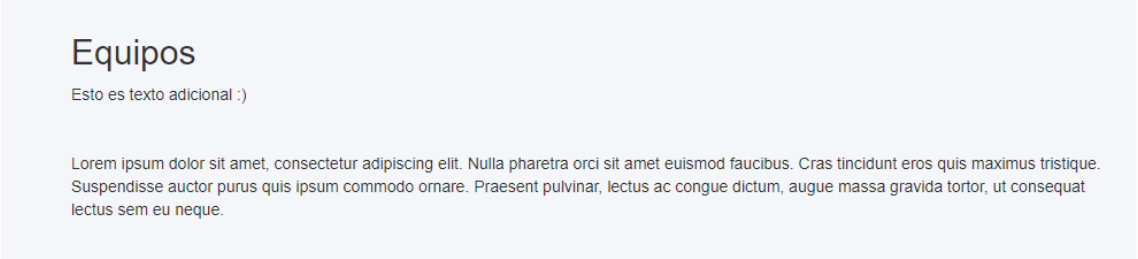

Figura 26 Contenido de la web 1a.

Figura 27 Modificación del orden del contenido.

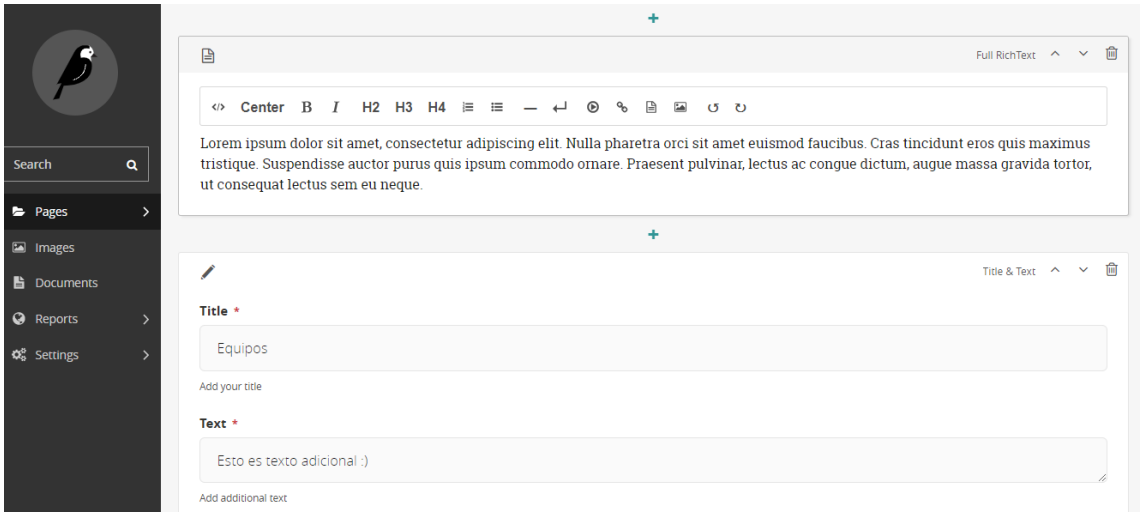

Figura 28 Acción de reordenación ya ejecutada.

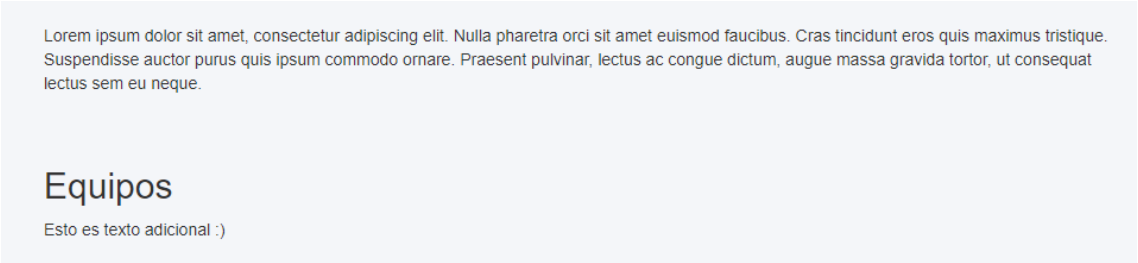

Figura 29 Contenido de la web 1b. Ya modificado.

Etiquetas y referencias en HTML:
Django, y por ende Wagtail, permiten referenciar las variables Python de las páginas dentro del código web HTML. Para ello se utilizan etiquetas.

```python
class TitleAndTextBlock(blocks.StructBlock):
    """Title and text and nothing else."""

    title = blocks.CharBlock(required=True, help_text="Add your title")
    text = blocks.TextBlock(required=True, help_text="Add additional text")
```

Figura 30 Variable en una clase en Python.

```html
<div class="container mb-sm-5 mt-sm-5">
    <div class="row">
        <div class="col-md-10 offset-md-1 col-sm-12">
            <h2>{{ self.title }}</h2>
            <p>{{ self.text }}</p>
        </div>
    </div>
</div>
```

Figura 31 Uso de una variable externa a través de funcionalidad Django.

Incorporación de una plantilla:
Para el cómo se ve la página web, se decide trabajar a partir de una plantilla. Para incorporar esta plantilla al CMS primero hay que conocer su estructura.

Para entender las bases, hay que conocer, por una parte, las carpetas contenedoras de las clases de páginas y por otra, la carpeta principal del proyecto.
Cada definición o tipo de página contiene una carpeta. El archivo con el que se trabaja de forma usual para gestionar el contenido de una página, es 'models.py'.
Este archivo contiene tanto los tipos de contenido de una página como el código HTML que extiende. El estándar para el trabajo es el uso de un HTML general para el proyecto y después un HTML específico para la página. Por ello se integra la plantilla de forma correcta a este archivo general, llamado 'base.html'.

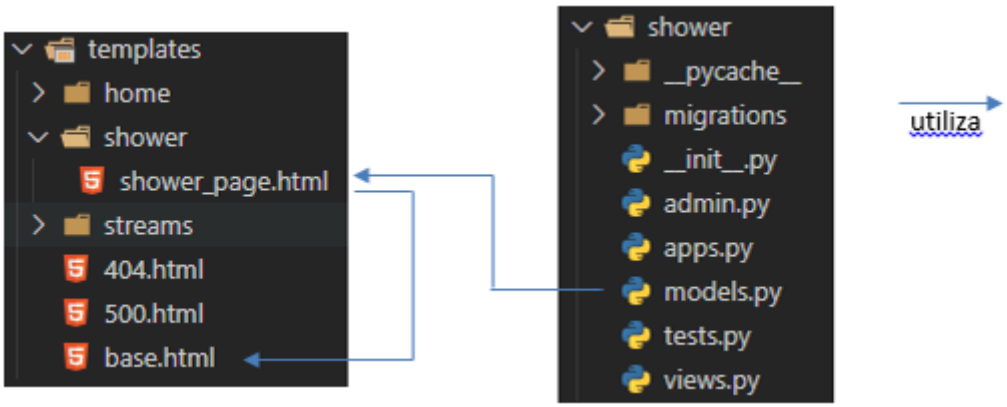

Figura 32 Estructura de uso HTML en una página en Wagtail.

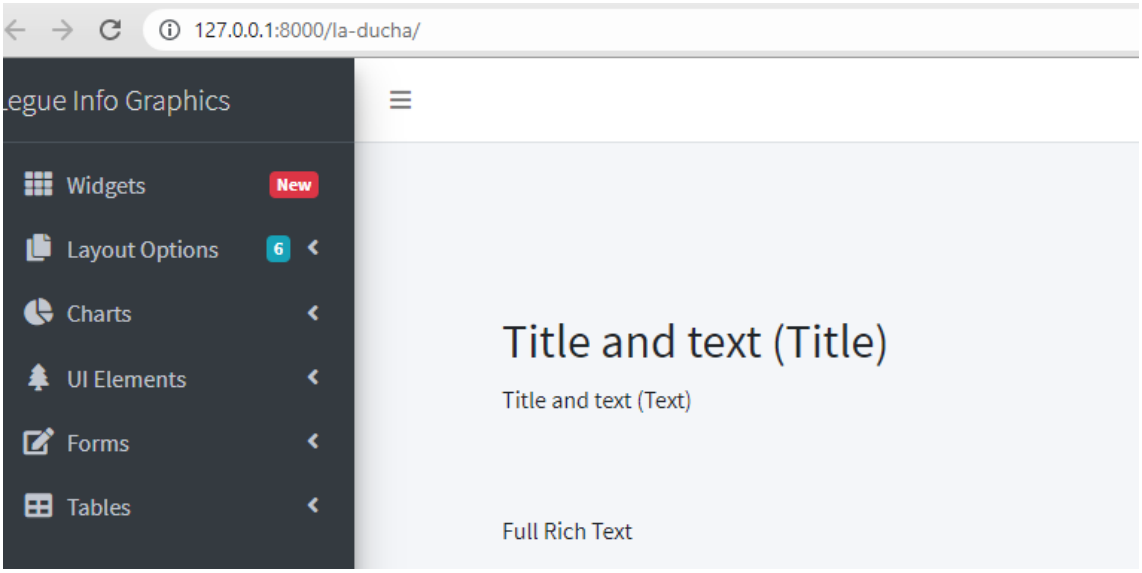

Figura 33 Plantilla acoplada al base.html.

Bloques personalizados:
Después se puede incorporar en el espacio del HTML específico el contenido de la página general. Pero, ¿cómo se genera contenido específico? Para esto existe una zona llamada 'block content', que es simplemente una etiqueta. Esto permite que la zona marcada en el archivo general pueda ser referenciada en el archivo específico. Gracias a esta funcionalidad se consigue incorporar un código relativo al tipo de página, de manera modular, a la interfaz generada por el CMS.

Figura 34 Zona de contenido específico de una página y zona donde se incorpora este contenido en el HTML base.

Para la creación de bloques personalizados hay que tener en cuenta el tipo de contenido que se desea crear. En la búsqueda sobre cómo implementar estos bloques personalizados, se han encontrado bloques ya creados con funcionalidades básicas. Estos no sólo permiten entender su funcionamiento sino que pueden ser usados para el TFG. Entre ellos se encuentran la inserción de texto, imágenes, links, etc. Uno de los puntos relevantes del tema es la estructura estandarizada para este tipo de datos (los bloques). Simplemente se separan en una carpeta aparte y se pueden agrupar en un archivo único.

Ahora para crear un bloque personalizado, que permita crear gráficas o tablas para un conjunto arbitrario de datos a elección, hay tratar dos temas:

a) **Como usar el módulo de procesado de datos (Conexión entre el CMS y el procesado).**
b) **Creación e incorporación de gráficas en el CMS.**

a) Como usar el módulo de procesado de datos (Conexión entre el CMS y el procesado).
El primero de los pasos es tratar la conexión entre el CMS y el procesado. Para ello existen dos posibles escenarios:

- **Encapsular la conexión del servicio del procesado en una tecnología**

Comúnmente para ofrecer un servicio web se utilizan las tecnologías Web API. Éstas permiten tener un sistema tanto optimizado para la gestión de las conexiones como para encapsular las funcionalidades para ser usadas por cualquiera y desde cualquier parte.

- **No encapsular la conexión del servicio del procesado en una tecnología**

También existe una solución más simple, donde las funcionalidades estarían integradas en el CMS/framework. Por una parte se pierde la encapsulación para el uso remoto público-privado, pero en la otra parte, se consigue una estructura eficiente en la gestión de la ejecución. En el caso de este sistema, seguiría manteniendo estas ventajas. Esto es debido a que las funcionalidades de procesado a usar, están encapsuladas en la estructura de api del propio CMS/framework, concretamente en forma da página.

Por simplicidad y carga de trabajo se ha escogido la segunda opción, dejando la posibilidad de cambiar la estructura en un futuro.

b) Creación e incorporación de gráficas en el CMS.
Una vez decidido qué se pretende incorporar el procesado dentro del CMS, el proceso para usarlo es bastante sencillo. Gracias a que ambas herramientas comparten la tecnología del lenguaje de programación (Python), simplemente el procesado se introduce como un nuevo paquete. Una vez incorporado, hay que conseguir que una 'page' utilice las nuevas

funcionalidades. Éstas serían obtener datos procesados a partir de parámetros y, obtener una gráfica a partir de estos datos procesados. Para ello se ha separado la resolución en dos puntos.

Primero, hay que buscar información sobre cómo utilizar variables derivadas de otras variables dentro del CMS. Específicamente, cómo usar una variable derivada en una 'page'. Esto se consigue creando una nueva variable del tipo 'property', dentro del modelo de la página, y llamándola desde HTML a través de su etiqueta asignada.

```
lpdb_ = GraphicProcessor(DataProcessor(DBMongo()))

def _results(self):
    data = self.lpdb_.data_proc.getTeamsBoWonByYearDataAccomulated('International',5)
    graph_name = self.banner_subtitle
    graph = self.lpdb_.createBarsGraph(data, graph_name)
    script, div = components(graph)
    result = script + div
    return result

graph = property(_results)
```

Figura 35 Creación de un valor derivado en una página tipo del CMS.

Segundo, y para incorporar las gráficas Bokeh se requiere de la incorporación de las bibliotecas para su uso HTML. La estructura de una gráfica Bokeh en HTML tiene dos componentes. El primero, es un código JS (Java Script) que genera la gráfica y contiene su información y, el segundo, es su componente HTML que contiene la referencia para utilizar este JS. En este caso, la respuesta que genera la gráfica contiene ambas partes (script y div) [7].
Pero al llamar a la variable ha surgido un problema. Después de investigar el motivo es que al incorporar texto externo a la parte HTML éste se le pasa de manera literal. Por motivos de seguridad, para evitar ataques XSS (Cross-site scripting) se previene que se ejecute cualquier funcionalidad dinámica de estos añadidos exteriores [8]. Para evitarlo y que la página web ejecute de manera corriente las funcionalidades del texto insertado, hay que añadirle en su llamada una etiqueta extra que permita esto. La etiqueta en cuestión es "safe" y necesita estar espaciada por un símbolo "|".

```
    </div>
    <!-- ./col -->
    {{ self.banner_subtitle }} <br>
    {{ self.graph | safe }}
</div>
```

Figura 36 Uso de una variable derivada en html.

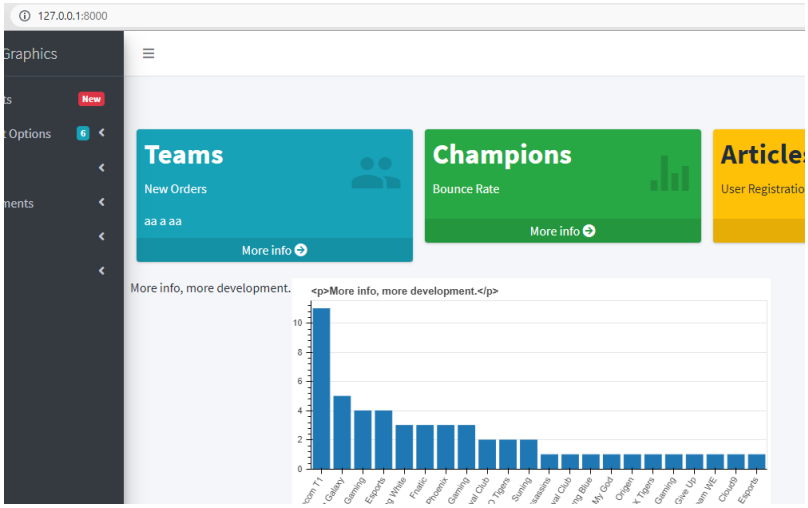

Figura 37 Gráfica incorporada directamente desde la plantilla de la página.

Cabe destacar que existen otros métodos para la incorporación de estas gráficas, como la inserción completa de toda la funcionalidad en un único componente, pero se ha optado por éste debido a su mayor versatilidad y modularidad.

Pero claro, lo que se pretende utilizar son bloques, por lo que de nuevo se ha investigado como utilizar variables derivadas aunque esta vez dentro de un bloque. Esto se consigue utilizando el contexto del bloque, que permite calcular nuevas variables asociadas al propio bloque. Este contexto se llama cuando se pide cualquier valor de la página. Estos valores están en forma de clave dentro de una variable “context” asociada.

```
data_source = blocks.ChoiceBlock(required=True, choices=DATA_SOURCE_CHOICES
region = blocks.ChoiceBlock(required=True, choices=REGION_CHOICES, help_tex
name = blocks.CharBlock(required=True, help_text="Add a name for your graph

data_proc = DataProcessor(DBMongo())
graph_maker = GraphicProcessor(data_proc)

def get_context(self, value, parent_context=None):
    context = super().get_context(value, parent_context=parent_context)
    data = getattr(self.data_proc, value['data_source'])(value['region'])
    graph = self.graph_maker.createBarsGraph(data, value['name'])
    script, div = components(graph)
    context['graph'] = script + div
    return context
```

Figura 38 Creación de un valor derivado (“graph”) en un bloque.

Figura 39 Generación de un contenido bloque tipo gráfica.

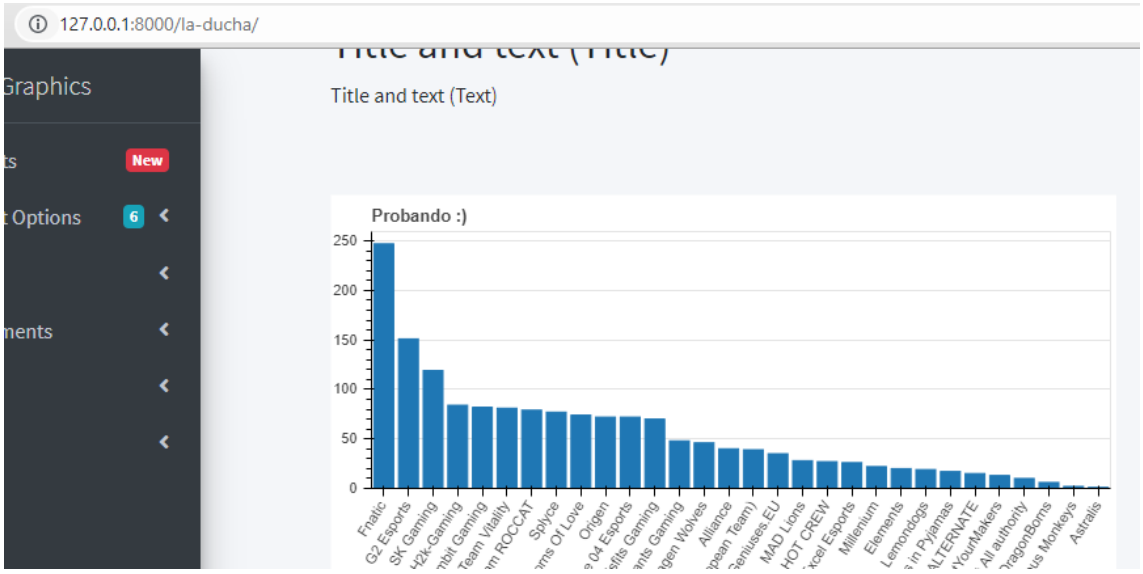

Figura 40 Gráfica incorporada como bloque.

Con esto ya se tiene la funcionalidad de cómo conseguir una variable que se calcula dinámicamente en tiempo de ejecución a partir de otras variables.

Cabe recalcar que este proceso de búsqueda de información, sobre el uso de variables procesadas dinámicamente, ha durado varios días. Esto es debido a que éste no es un tema sencillo que requiere de algunos conocimientos avanzados. Por lo que el trabajo de búsqueda sobre esta información se ha ido intercalando con el 'debbugeo' de otras partes del proyecto y con la incorporación de las gráficas en el HTML.

Con estos dos puntos se ha conseguido que un usuario gestor del sitio web, sea capaz de crear contenido de una gráfica de manera sencilla.

6.5.- Trabajo mixto

Redefinir estructura del contenido del proyecto para CMS.

Después de plantear el tema de cómo integrar el procesado en el CMS, se ha visto que el planteamiento sobre el objetivo del servicio que se genera con la aplicación no está del todo bien planteado. Es decir, que se estaba planteando la implementación de varias funcionalidades para uso de desarrolladores y no de gestores de contenido. De esta manera se replantea el uso de las funciones, pasando a preparar las funcionalidades para un uso no directo, que requiera de un manejo de datos interno, para personas sin conocimientos de desarrollo de aplicaciones.

Otro de los cambios presentes es la transición de utilizar gráficas Bokeh a gráficas con chart.js y DataTables. Esto permite por una parte que la estructura del proyecto sea más afín a la estructura planteada, separando más la generación visual de la información del procesado de datos. Por otra parte, se fomenta la integración de estas representaciones visuales con las tecnologías y herramientas que se utilizan en la parte de interfaz web. Esto se debe a que los dos paquetes a utilizar se usan en la parte JS del TFG. También permite que se aumente la calidad visual del sitio web.

La nueva implementación para chart.js tiene similitudes con Bokeh pero con un carácter más sencillo y directo. Para crear un gráfico en chart.js, por una parte, hay que asignarle las propiedades del gráfico y por otra los datos. Esto hay que implementarlo dentro de las características de un bloque. También hay que usar los datos del bloque de la página y no olvidarse de importar las bibliotecas de chart.js.

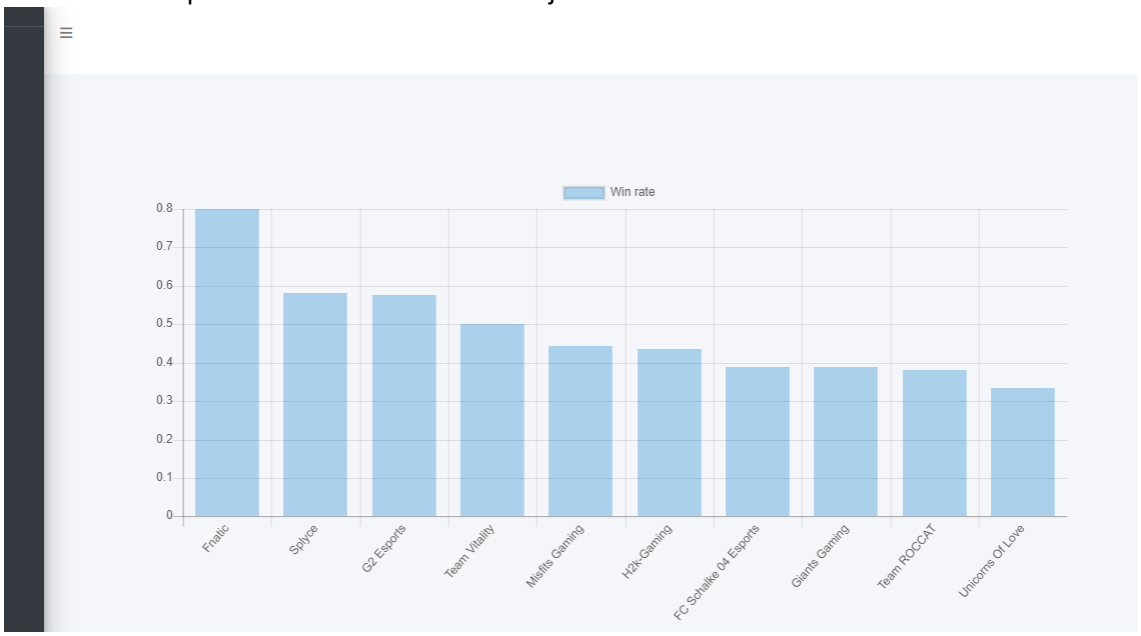

Figura 41 Gráfica generada con chart.js a través de un bloque.

Uso de paquetes externos.

El CMS en uso, Wagtail, tiene una comunidad no muy grande pero si activa. Esto significa que también tiene una cantidad suficiente de paquetes creados por la comunidad que añaden todo tipo de funcionalidades. En el caso de este TFG se han considerado el uso de dos paquetes: “wagtailuiplus” y “wagtailmenus”.

Uno de los puntos que se pueden mejorar de la parte de administración, es la interfaz a la hora de editar una página, ya que no dispone de una opción para colapsar o minimizar conjuntos de

bloques. Si tienes más de cuatro bloques en una misma página, cosa que no es muy extraño, hay que hacer un pequeño recorrido para ir del primero al último.

Por ello se busca un paquete que solucione este posible inconveniente. El paquete encontrado se llama 'wagtailuiplus'[9] y, entre otras cosas, permite el colapso de bloques. Pero una vez instalado y acoplado al CMS, no funciona del todo correctamente. Es debido a la versión de Python que utiliza. Este punto se puede arreglar, siempre que se instale una versión más antigua del CMS que aún utilice Python 2.x . Debido a que los inconvenientes de usar tanto una versión del CMS tan vieja como el uso de un Python viejo son demasiados (incluso hay funcionalidades creadas que dejan de funcionar), se decide mantener la versión actual. Esto no quita que se siga utilizando el paquete y qué alguna funcionalidad siga aún activa.

Otro paquete a utilizar es 'wagtailmenus'. Éste permite generar los menús desde el panel de administración, dando al gestor de contenidos una mayor versatilidad. Ahora se pueden referenciar menús, introducidos desde el panel de administración del CMS en el código del proyecto. De esta manera se pueden colocar en sus lugares adecuados.
Para ello hay que saber que la variable que los alberga se llama "main_menu". Hay que darle una plantilla HTML, sino utilizará la plantilla main_menu.html situada en la carpeta de plantillas "templates". Si no existe ninguna plantilla con ese nombre pues simplemente incorporará una lista de los menús con sus referencias asociadas.

```
{% main_menu template="menus/main_sidemenu.html" %}
```

Figura 42 Referencia a la variable del paquete 'wagtailmenus'.

La variable main_menu contiene menu_items. Éste a su vez está compuesto por una lista de nombres más referencias que tiene asociados el nombre de text y href. Con estos se puede indicar tanto el nombre como el enlace a la página asociada a su nombre.

```
{% for item in menu_items %}
    <a href="{{ item.href }}" class="nav-link">
        {{ item.text }}
    </a>
{% endfor %}
```

Figura 43 Uso de wagtailmenus en el código.

Una vez colocadas las referencias en el código del proyecto, sólo faltaría añadir estos menús desde el panel de administración del CMS.

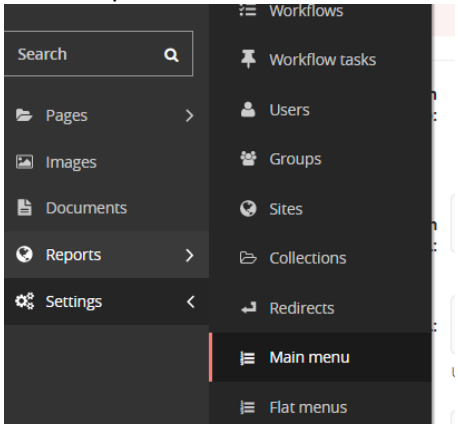

Figura 44 Nueva zona en ajustes para el trabajo con menús.

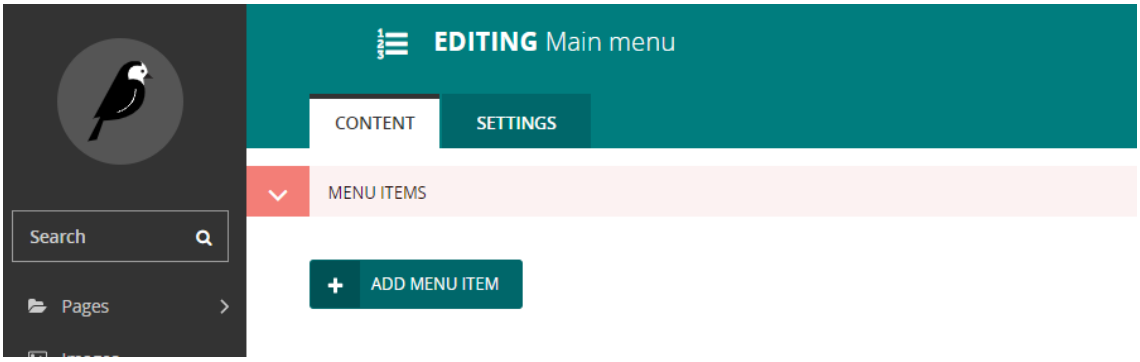

Figura 45 Zona de gestión de los menús.

Por último, hay que indicar si las páginas que se quieren utilizar se pueden usar en los menús del paquete. Para ello existen dos formas. Una en la que desde el panel de administración del CMS se decide por cada página. Y otra, donde se define en el tipo de una página el valor asociado a poder ser mostrado.

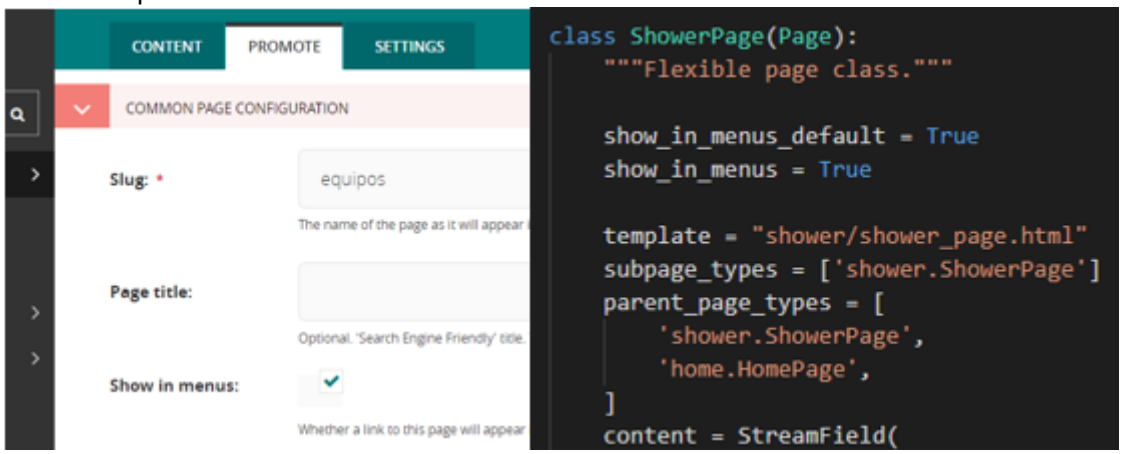

Figura 46 Incorporación de páginas a "wagtailmenus". Dos formas de permitir su uso.

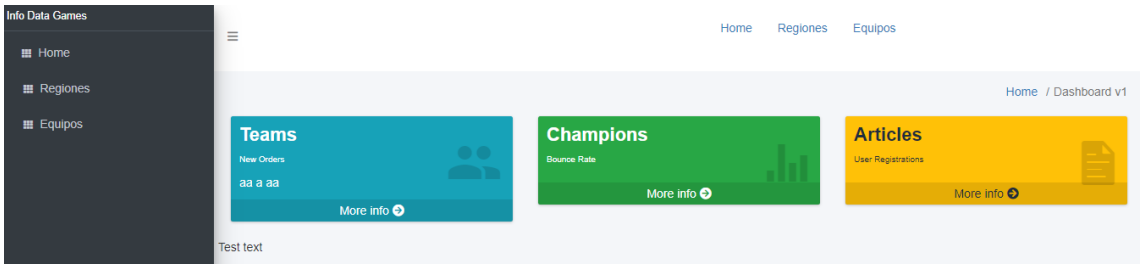

Figura 47 Menús acoplados a la barra lateral y a la cabecera de la página.

Dropdowns acoplados:
Una de las principales ventajas de utilizar estas bibliotecas de JS, es la posibilidad de acoplar nuevas funcionalidades asociadas con la web a las gráficas. En este caso se pretende añadir una barra desplegable de opciones para seleccionar entre diferentes gráficas. De esta manera se crean los elementos HTML y después, se añaden varias funciones JS que permitan transicionar de un conjunto de datos a otro, para acabar colocándolos en la gráfica.

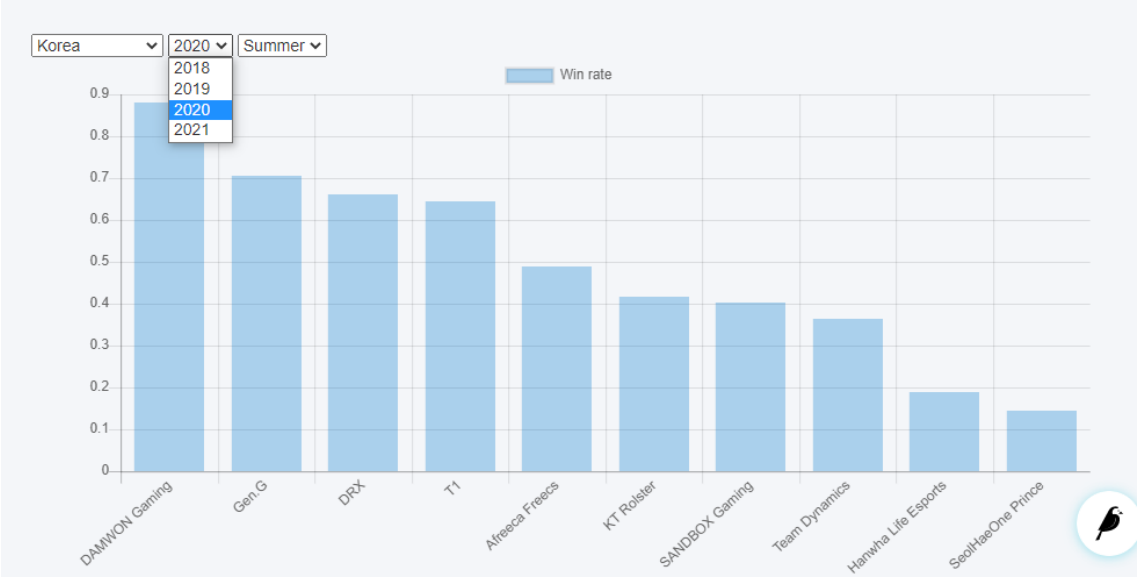

Figura 48 Gráfica con dropdowns asociados.

Estructura de carga de datos (carga de todo los datos):
Hasta ahora los datos procesados se están calculando cada vez que se requiere su uso, y esto está dando un problema para los tiempos de cálculo para algunos procesados pesados. Es decir, no es viable que un usuario tenga que esperar diez segundos para que la página web cargue. Es aquí donde vienen los planteamientos de posibles soluciones a través de diferentes estructuras:

	Mem fisica			Mem virtual			cpu			ttotal
	DB	WG	MOD	DB	WG	MOD	DB	WG	MOD	
POST(3->4)	=	-	+	=	-	+	+	-	+	~=,(+)
OPERATION_ON_BD	=	=		+	-		++	--		~=,(-)
DB_PROCESSED	+++	+		=	-		=	-		-

Figura 49 Opciones de optimización del tiempo de búsqueda más procesado.

- **POST (SI de tres a cuatro capas)**
  Pros: El sistema permite reducir los tiempos para algunos escenarios. Entre estos escenarios la carga dinámica de equipos, donde se gana carga inicial por carga continua. Es decir, escenarios disjuntos.
  Contras: El código gana cierta modularidad, no necesaria para la envergadura del proyecto, ya que el procesado es un servicio que sólo consume esta aplicación. El

sistema gana carga de trabajo continuo. Se requieren más aspectos técnicos como el manejo de URLS a través del CMS.

- **Mejorar las operaciones en la base de datos (querys)**
  Pros: Se ganaría un porcentaje significativo de eficiencia en los tiempos de cálculo totales.
  Contras: Trabajo cerrado y con cierto grado riesgo/recompensa. En casos de escenarios muy conjuntos, el rendimiento del servidor de la BD podría verse afectado.

- **Crear una base de datos procesados**
  Pros: Se ganaría la máxima eficiencia posible, solo dependiente del tiempo de consulta a la nueva base de datos.
  Contras: La memoria física de la BD aumenta de manera exponencial.

- **Punto intermedio**
  Idea desechada: Se podría implementar un punto medio, donde se guarden los datos por el tipo de procesado (su agrupación) y al cargar haya un ligero filtrado. Esto requiere mayor planificación, complejidad, pérdida de modularidad y tiempo de trabajo del proyecto.

La solución escogida es la creación de una nueva base de datos procesados. Y es que la cantidad de contenidos con datos procesados en uso al mismo tiempo, en la práctica, sería poco significativa. Es decir, la web igual contiene como mucho cincuenta o cien gráficas al mismo tiempo reduciendo el mayor problema de éste. Por otra parte este punto se acerca más a las metas de crear un SI lo más completo posible.

Manager.
Entonces, para llevar a cabo los cambios propuestos hay que guardar los datos procesados. Para almacenar los datos procesados, lo primero es recordar la segunda estructura propuesta para el sistema.

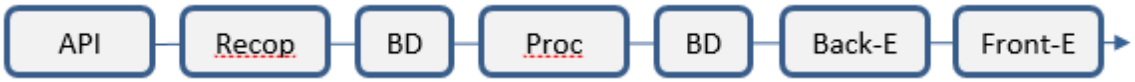

Figura 50 Secuencia de diseño mejorada.

Como se puede observar, la idea principal es que el flujo de los datos se separa entre el CMS y el procesador. De esta manera lo que se pretende crear es un gestor del módulo de procesamiento. Este módulo se decide llamar "manager_model". ¿Cuáles son sus funciones? Por una parte, necesita recibir peticiones para obtener datos ya procesados y responder con éstos. Por otra parte, debe de poder actualizar los datos, ya procesados, que deriven de datos sobre torneos en marcha (aún activos). Para ello se han planteado dos estructuras posibles.

- **Estructura A:**
  La parte de obtener los datos y actualizarlos en una misma función.
  Sólo actualiza datos necesarios.
  Requiere de comprobaciones constantes.
  Más sencillo de implementar.

- **Estructura B:**
  La parte de obtener los datos separada de actualizar los datos.

Permite separar funcionalidades sin cargar más el sistema.
Requiere de más configuración.
Más complejo de implementar.

Debido a los plazos y la previsión de tiempo se decide optar por la opción A con posibilidad de transición a la opción B.
El desarrollo de este módulo ha constado de tres fases:

1. *Diseño para un caso único*
2. *Transposición a una Estructura B*
3. *Generalización del funcionamiento*

1.- Diseño para un caso único
La idea principal es que se requiere guardar alguna referencia de los datos entrantes para poder diferenciar que datos de la base de datos obtener. En este caso está claro, que los parámetros de la función (y que tipo de procesado tiene, su fuente, que se tratará luego) es la mejor referencia. Esto permite poder tener una referencia directa de uso externo al sistema con el dato procesado que se intenta obtener. Por otra parte y si se pretenden actualizar los datos, también se necesita potencialmente la fecha de actualización. De esta manera se añaden a los datos procesados dos bloques de campos extras.

Ahora para la obtención, creación y actualización se consideran los siguientes casos:

- Existen los datos a obtener
  - ➔ Los datos están actualizados: Se entregan los datos ya existentes
  - ➔ Los datos no están actualizados: Se calculan nuevos datos
- No existen los datos a obtener: Se calculan nuevos datos

```
.Estructura Mixta
if datos
    if datos not in torneoactivo
        return datos
    else if datosactualizados
        return datos
    else
        datos = calcularNuevosDatos()
        datos[fecha] = fecha.ahora
        return datos
else
    datos = crearNuevosDatos()
    datos[fecha] = fecha.ahora
    return datos
```

Figura 51 Recorrido para obtener datos procesados en el manager v1.

También hay que tener en cuenta que el manager requiere de acceso a las dos bases de datos (base y procesados), donde los datos procesados se colocan en una nueva base de datos. Exactamente en una colección asociada a su tipo de procesado.

2.- Transposición a una Estructura B
Una vez implementada la funcionalidad y visto que el cambio de ésta es bastante sencillo, se transpone la parte que actualiza "al vuelo" a otra función. Ésta permite actualizar los datos y puede ser llamada cuando se necesite o quiera. El principal desafío encontrado es recorrer todos los documentos de todas las colecciones de los datos procesados y, no se ha encontrado ningún problema relevante.

```
.Estructura Separada
-Obtener Datos
if datos
    return datos
else
    datos = crearNuevosDatos()
    datos[fecha] = fecha.ahora
    return datos

-Actualizar Datos
for coleccion in base_de_datos
    for documento in coleccion
        if documento in torneoactivo
            datos = calcularNuevosDatos()
            datos[fecha] = fecha.ahora
```

Figura 52 Separación de funcionalidades.

3.- Generalización del funcionamiento

En este punto se pretende modular lo más posible el funcionamiento de estas dos partes del manager. Ya no se quiere que el manager actúe para una función en concreto, sino para cualquier función.

Para ello, primero, hay que hacer que las funciones traten como input una cantidad cualquiera de parámetros. Y segundo, añadir la función de procesado origen de donde se pretende obtener los datos.

- Cantidad variable de parámetros:
  En este caso interesa el tratamiento de parámetros con nombre. Para tratar cualquier número de parámetros con nombre se utiliza el doble asterisco, que permite obtener los parámetros como una lista de tuplas clave-valor. Esta lista de tuplas tiene la estructura de un diccionario de Python, por lo que es fácil de usar. Es decir, como argumento de la función se tiene '**params'. Ahora la variable 'params' puede ser usada de manera sencilla. Claro está que si la idea es que sirva como parámetro de cualquier función, este diccionario se tiene que poder pasar como un conjunto de argumentos. Para ello simplemente hay que referenciar de la misma manera a la variable. Por lo que al introducir como parámetro un diccionario en forma '**dict' o al introducir el conjunto de parámetros obtenidos en forma '**params' ('params' que también tiene estructura de diccionario) se consigue que una única variable compleja se convierta en un conjunto de parámetros de entrada.

- Función de procesado variable:
  Aparte de poder pasar cualquier tipo y cantidad de parámetros se necesita la función objetivo que utilizará a estos. Para ello se ha decidido añadir en los datos procesados su origen, llamado en la bd 'source'. De esta manera se puede obtener el origen directamente de los datos ya procesados. En el caso de crear los datos se necesitará esta variable origen. Para tratar de manera más legible y modulada se decide crear un documento 'config', que relaciona un nombre casual con el nombre de la función específica del procesado.

```
def getPorcFunction(self, name_id):
    name_function = proc_data_config.FUNCTIONS[name_id]
    return getattr(self.data_procesor, name_function)
```

Figura 53 Transformador de id a función.

No cabe duda de que al obtener los datos procesados, los campos creados para su mantenimiento y gestión son eliminados de esta respuesta.

Otros desafíos en la vista.

Uno de los principales problemas encontrados es a la hora de insertar dos gráficas del mismo tipo en la misma página. Esto se debe principalmente a que al insertar el código HTML+JS de un bloque a la página, se añade de manera literal. De esta manera se crean referencias del código no encapsuladas o mal definidas que se comparten entre los dos bloques de gráficas. Para solucionar este problema los arreglos son los siguientes:

a) Cambios en las referencias HTML:

Para estos casos se añaden a sus ids un componente extra dependiente del bloque. Para simplificar se ha decidido añadir el nombre de la gráfica al id. Esto permite que el código sea similar (en su punto de vista legible) aunque tiene el problema de que si existen dos gráficas donde el usuario les da el mismo nombre, daría algún problema de id. En un caso práctico es muy poco probable que se dé este problema. De momento se deja como problema a solucionar más adelante.

b) **Referencias en JS:**

La solución en este punto es clara, encapsular la funcionalidad creada en una clase. A la hora de realizar esta solución se han encontrado dos desafíos.

i) Privacidad de las variables:

A la hora de usar variables de clase dentro de varias de las funciones internas, se ha encontrado un problema. Estas variables no eran leídas de manera adecuada y se trataban como si no tuvieran un valor asignado en alguna parte. Esto es debido al uso de funciones con 'callbacks', es decir, funciones con otra función como parámetro. El caso es que estas nuevas funciones 'callback' creadas al vuelo no están dentro del ámbito de la clase y no pueden obtener la información de estas variables. Por lo que se parametrizan estos parámetros solucionando el problema.

```
function updateChart() {
    jsonactive = getDropdownIDValues();
    jsonactive.sort((a, b) => parseFloat(b[fi

    labels = jsonactive.map(function(e) {
        return e['team'];
    });
    data = jsonactive.map(function(e) {
        return e[field];
    });
    chart.data.datasets[0].data = data;
    chart.data.labels = labels;

    chart.update();
}
```

```
var activefield = this.field;
var data = jsonactive.map(function(e) {
    return e[activefield];
});
```

Figura 54 Solución alternativa para una función callback.

ii) Declaraciones de clases ya declaradas:

El problema principal es que el código JS está colocado en el HTML, por lo que al declarar las clases se solapan y da algún error. Para evitar que la página se bloquee, se definen estas gráficas en archivos externos de JS y luego se importan.

Para utilizar clases locales importadas, hay que indicar que se van a utilizar archivos estáticos y después referenciarlos con la etiqueta de Django adecuada.

```
{% load static %}
<script type="text/javascript" src="{% static 'js/bars_graph_vertical.js' %}"></script>
```

Figura 55 Carga de archivos JS de manera estática.

Una vez encapsulada la funcionalidad de creación de la gráfica, ya se puede instanciar cualquier cantidad del mismo tipo sin ningún problema.

Base de datos remota.
Para la conexión remota la tecnología usada es Atlas. Esta es una base de datos NoSQL remota y con posibilidad de uso gratuito. Una vez registrado, para utilizar esta base hay que seguir los siguientes pasos:

1. Crear un cluster
2. Añadir permisos
3. Integrar la conexión de la base de datos al código

6.6.- Despliegue

En la búsqueda de opciones para el despliegue se han encontrado diferentes alternativas:

a) **Despliegue manual:**
En este caso se configura e implementa todo el despliegue de manera manual. Entre las tareas a hacer estarían la obtención del dominio, de un servidor, de la configuración de la máquina obtenida y del acoplamiento de la aplicación. La principal ventaja sería el control que se tiene sobre el despliegue, pero el inconveniente es la carga de trabajo y el riesgo añadido de encontrarse nuevos problemas.

b) **Despliegue a través de un servicio en la nube:**
Hoy en día existen bastantes plataformas como servicio, también conocidas como 'PaaS' (Platform as a Service), que permiten subir una aplicación personalizada a un entorno con ciertas configuraciones del proceso de despliegue ya hechas. En concreto, la página oficial de Wagtail da a conocer tres 'PaaS': PythonAnywhere, Heroku y Digital Ocean. De entre estas opciones destaca Heroku por dos motivos: existe una documentación más activa y permite la integración de un proyecto Django/Wagtail de forma directa.

c) **Otros puntos intermedios:**
Por último, comentar que también existen otros puntos intermedios entre ambas opciones donde se relega alguna funcionalidad al proveedor. Un ejemplo sería Google Cloud que actúa como una infraestructura como servicio. En este caso se facilitaría el proceso de obtención de dominios y servidores.

A continuación, se va a proceder a explicar cómo integrar un proyecto de Wagtail en Heroku, la opción elegida para este TFG.

Para asegurar la evasión de posibles fallos de configuración, se recomienda primero la creación de un proyecto en blanco y después acoplar el proyecto propio.

1. **Instalación de pipenv**. Comúnmente para trabajar con proyectos Python que utilizan paquetes o librerías externas se utiliza pipenv. Éste es un gestor de dependencias.

2. **Crear el espacio de trabajo**. Para ello se debe configurar un entorno pipenv e incorporar en este entorno las librerías que se van a utilizar. Entre ellas, por ejemplo, se incluye Wagtail (pip install wagtail dentro del entorno).
3. **Reconfigurar el proyecto Wagtail**. Para que Heroku lea y ejecute, de manera adecuada, se han de cambiar varios puntos de configuración del proyecto.
   A) **Cambiar la bd**: Wagtail utiliza, de manera predeterminada, como gestor de base de datos 'SQLite'. Aunque este gestor es perfecto para pruebas y pequeños proyectos se recomienda cambiarlo. En este caso se va a instalar PostgreSql ya que éste es el gestor predeterminado de Heroku.
   Primero se modifica la indicación de que base de datos utiliza el proyecto, después hay que instalar y crear la nueva base.

   B) **Cambios en las configuraciones para la interpretación de Heroku.** Entre ellos se encuentran ediciones y añadidos. Cambios en las diferentes configuraciones y el uso de WhiteNoise, como paquete, que permite optimizar el uso de archivos estáticos.

4. **Despliegue da la aplicación.**
   El primer paso es registrarse en la página de Heroku. Para el despliegue se requiere la tecnología de git. Los pasos a seguir son los siguientes [10].

   - Añadir el contenido a la rama local actual de git.
   - Generar una nueva versión del proyecto git con la rama local.
   - Crear un proyecto Herorku nuevo.
   - Subir el proyecto git a Heroku.
   - Actualizar configuraciones en el proyecto.
   - Migrar el proyecto recién subido de Heroku.
   - Crear un usuario administrador en el proyecto de Heroku
   - Asegurar la asignación de recursos de Heroku al proyecto

Una vez realizados todos estos pasos, el proyecto se encuentra ya funcional. Un último retoque sería cambiar el nombre del dominio. Para ello se puede utilizar la interfaz web de Heroku o actualizarla por la consola de comandos con “heroku apps:rename newname”.

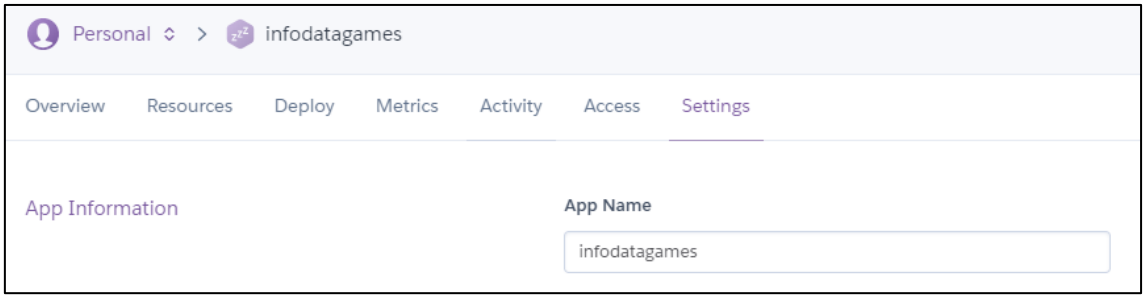

Figura 56 Panel de administración de un proyecto Heroku.

7.- Pruebas

Para la realización de las pruebas se han utilizado diferentes herramientas online que detectan los posibles errores de una página web, en relación con la accesibilidad, subcaracterística del requisito de usabilidad presente en la norma ISO 25010 [11] de la calidad de producto software.

Figura 57 Contenido de la página web sometida a las pruebas.

TAW es una página de pruebas especializada en la accesibilidad que se basa en las pautas WCAG 2.0 [12]. Se han realizado las pruebas a Nivel AAA.

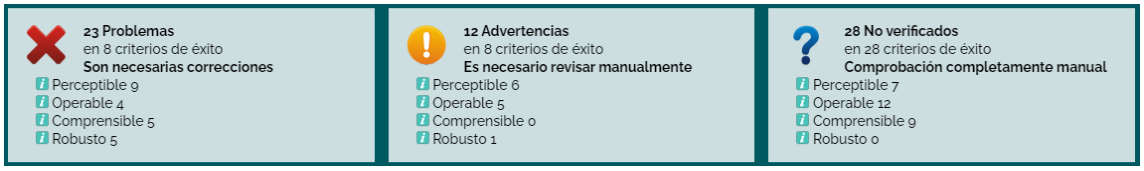

Figura 58 Resultados de TAW.

Los principales problemas encontrados es un contenido no textual, es decir, que falta texto. Esto se debe, por una parte, a que es una página de pruebas y no contiene mucho contenido. Por otra parte, el carácter de la página no incita a colocar mucho texto en ella.

Otros problemas son el título de “Info Data Games” que actúa como un link, pero no tiene aún asociado ningún enlace y que el idioma de la página no es variable ni está modulado.

Validador WC3 está especializado en la validación de marcado del contenido web.

Figura 59 Resultados de Validator WC3.

Los únicos errores detectados por Validator tienen relación con la indicación de etiquetas especializadas del framework/CMS. Estas etiquetas hacen referencia al contexto que se utiliza en django dando que el análisis no sea capaz de interpretarlo. Es decir, que el type="…" marcado en las etiquetas son necesarias para definir el tipo de datos.

Examinator se centra también en los puntos dictados sobre la accesibilidad en el WCAG 2.0.

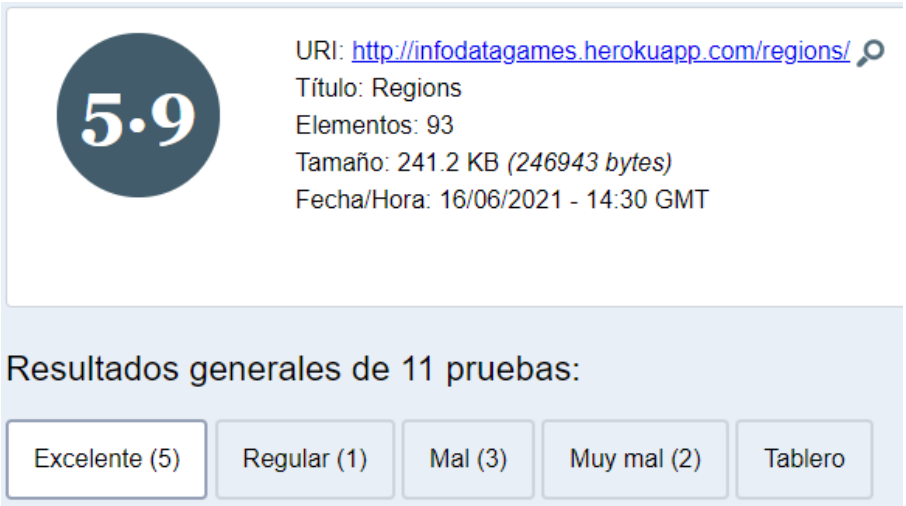

Figura 60 Resultados Examinator.

Los principales errores encontrados son el uso de enlaces no correctos. Primero, está el problema comentado en un punto anterior donde el nombre de la página no lleva a ninguna parte. Un segundo punto es el problema de que existan dos enlaces que llevan al mismo lugar. También se comenta que faltan enlaces extra para saltar los bloques de contenido de la página.

En los puntos positivos se valora la indicación del idioma en el código, que éste utiliza encabezados y títulos, que se utilizan los elementos semánticos para marcar la estructura y que la presentación del contenido está suficientemente separada para que sea distinguible.

HERA se especializa en una versión anterior del WCAG, en concreto en la versión 1.0.

Status of checkpoints

Priority	Needs checking	Pass	Fail	N/A
P1 HERA WCAG 1.0	4	1 ✔	--	12
P2 HERA WCAG 1.0	7	9 ✔	4 ✘	9
P3 HERA WCAG 1.0	11	--	2 ✘	6

Figura 61 Resultados HERA.

Hera es capaz de encontrar los diferentes problemas sobre los enlaces y sobre la estructura como el no uso de headers y otros elementos como una mejor estructuración en bloques.

Los puntos positivos encontrados son los nulos problemas que generan los JS. Por otra parte, los elementos de estilo están separados en su propio documento, la carga de la página es suave y que la página no utilice ningún script que recarga de manera periódica su contenido son puntos a favor. Por finalizar, otro punto bueno, comentado es el no uso de etiquetas HTML desfasadas

8.- Conclusiones

Desde un punto personal, este TFG ha sido un desafío. Por una parte el alumno se enfrenta a un problema con una envergadura nunca tratado y con cierto toque solitario. Los atributos que más pueden llegar a destacar, en este tipo de trabajo, son la adaptabilidad ante situaciones, la determinación para la toma de decisiones y la constancia para el desarrollo del proyecto.

Para la realización de este han sido clave varios puntos.

- **Definición de problemas concretos:**
  En proyectos de una mayor envergadura es muy difícil para una persona tanto recordar de manera activa todos los temas aplicados como planificar a través de grandes objetivos. Reducir la envergadura, sobre el planteamiento de los problemas, y enumerarlos de manera adecuada ha sido clave para el desarrollo relativamente fluido del proyecto.

- **Aprendizaje rápido:**
  El aprendizaje rápido es necesario para proyectos con un enfoque más multidisciplinar. En el desarrollo de este proyecto se ha aplicado este método, siempre y cuando ha sido posible. Es decir, aprender los temas y puntos concretos sobre tecnologías y herramientas no conocidas intentando evitar el aprendizaje de todas las funcionalidades colindantes al tema buscado. También hay que tener en cuenta que este tipo de aprendizaje no se da sobre el uso de herramientas con conocimientos ya existentes.

- **Confianza en resultados a largo plazo:**
  Una cosa está clara. Se trabaja con el proyecto tanto con el mayor tamaño tratado como con el desarrollo más longevo. Esto implica que existen ciertas dudas y desconfianzas sobre el resultado, qué va a quedar hecho una vez acabado el proyecto. Es muy difícil de estimar qué se debe hacer exactamente y si los pasos dados son los correctos. De esta manera, la necesidad de confiar en los resultados futuros es fundamental.

La mayoría de los problemas tratados tienen relación con el conocimiento sobre el contexto de los tecnicismos o por la falta conceptual sobre éstos. En concreto, las diferentes opciones para la fuente, la tipología de datos, los funcionamientos específicos sobre tecnologías o la búsqueda y planteamiento sobre las opciones para la resolución de problemas.

Eso sí, los puntos negativos no quitan que la realización del proyecto haya sido una experiencia muy positiva que ha ayudado a familiarizarse con un entorno de trabajo mucho más práctico. Sin olvidarse de que se ha ganado cierta preparación para la estructuración de proyectos completos. Todo esto permite tener una gran preparación para enfrentarse al mundo laboral, ya sea de manera individual o de manera colectiva.

9.- Trabajo Futuro

Dentro del trabajo futuro se puede encontrar un desempeño individual o colectivo.

El desempeño individual vendría a ser el trabajo autónomo ya sea sobre la aplicación o sucedáneos. Es decir, ya conseguidas unas bases se podría partir del mismo sistema mejorando algunos puntos como la separación del procesado sobre el CMS, el uso de nuevas fuentes, la mejora sobre Heroku o, sin olvidarse, de la ampliación de los tipos de contenidos ya existentes como datos procesados y tipos de vistas.

Para proyectos de mayor escala, con un trabajo más colectivo, se podrían utilizar los conocimientos adquiridos a través del TFG. En este saco entrarían temas como el desarrollo de un sistema de información, el uso de tecnologías web relacionadas con Python o la generación de nuevos datos derivados.

10.- Licencia Software y Documental

Llegados a este apartado, se va a proceder a comentar tanto la licencia de software como la licencia documental.

En cuanto a la licencia de software se va a emplear Berkeley Software Distribution (BSD). Se trata de una licencia de software libre permisiva como puede ser OpenSSL o la MIT License. Existen diferentes tipos de licencias, en el caso de este TFG se ha utilizado la licencia "BSD modificada", "BSD revisada", "BSD-3" o "BSD de 3 cláusulas" [13].

Figura 62 Logotipo de la licencia BSD.

Al igual que sucede en el mundo del software, se tienen que buscar formas de garantizar las libertades asociadas al trabajo elaborado y su inviolabilidad futura. Para garantizar que la libertad esté asociada al documento se buscan métodos, uno de ellos es la licencia GNU Free Documentation License (GFDL).

Figura 63 Logotipo de la licencia GNU.

El propósito de esta Licencia es hacer que en el caso de este TFG sea "gratuito" en el sentido de libertad: para asegurar a todos la libertad efectiva de copiarlo y redistribuirlo, con o sin modificarlo, ya sea comercial o no comercialmente. En segundo lugar, esta licencia preserva para el autor y el editor una forma de obtener crédito por su trabajo, sin ser considerado responsable de las modificaciones realizadas por otros. Es una especie de "copyleft", lo que significa que las obras derivadas del documento deben ser libres en el mismo sentido.

Si por algún motivo se emplea este documento y se modifica, debe realizar una serie de acciones indicadas en el sitio web oficial de GNU [14].

Tampoco hay que olvidar que este documento, por defecto, está al amparo de la 

, por su inclusión en el Repositorio Institucional de Documentos de la Universidad de Zaragoza: ZAGUAN.

11.- Referencias Bibliográficas

[1] https://www.lineups.com/esports/top-10-esports-games/
[2] https://www.ganttproject.biz/
[3] https://www.ctr.unican.es/asignaturas/is1/IEEE830_esp.pdf
[4] https://lol.fandom.com/wiki/Help:Leaguepedia_API#Limits
[5] https://docs.bokeh.org/en/latest/docs/reference/palettes.html
[6] https://www.youtube.com/watch?v=GyULF8HzreI&ab_channel=CodingForEverybody
[7] https://edx.readthedocs.io/projects/edx-developer-guide/en/latest/preventing_xss/preventing_xss_in_django_templates.html
[8] https://docs.bokeh.org/en/latest/docs/user_guide/embed.html#components
[9] https://pypi.org/project/wagtailuiplus/#description
[10] https://github.com/CodingForEverybody/wagtail-heroku-deployment#applying-git
[11] https://iso25000.com/index.php/normas-iso-25000/iso-25010
[12] https://www.w3.org/TR/WCAG20/
[13] The GNU Operating System and the Free Software Movement, 2021. Gnu.org [online]
[14] Licencia BSD - Wikipedia, la enciclopedia libre, 2021. *Es.wikipedia.org* [online]

12.- Índice de Figuras