1.- Introducción:

Desde hace un tiempo hasta ahora nuestra sociedad se ha visto sacudida por varios escándalos en cuanto a la interpretación de los sucesos que ocurren en todo el mundo. No hablamos de las tensiones políticas que puedan existir entre países, o problemas gravísimos como las situaciones que se están viviendo en algunos países de Oriente Medio. A lo que nos referimos es a la manipulación de la información, a cómo ciertos medios pueden llegar a tergiversar cierta información para su beneficio o simplemente por el hecho de tener a la población “engañada”.

Este fenómeno ha existido toda la vida, en mayor o menor medida, pero quizás es hoy en día cuando tiene un impacto mucho mayor en nuestro día a día, por el efecto de Internet y las redes sociales, que magnifican todo mucho más. Tenemos ejemplos de esta manipulación y como ha llevado incluso a sugestionar a ciudadanos para distintos intereses, algunos ejemplos de ello son [el impacto de las "fake news" en las elecciones de EEUU](https://www.lavanguardia.com/tecnologia/20190408/461469794060/fake-news-propaganda-desinformacion-trump.html), en las [elecciones brasileñas](https://www.elespanol.com/mundo/america/20181016/fake-news-inundan-campana-electoral-brasil/345716563_0.html), y no sólo en este campo afectan este tipo de bulos, si no que también a otros aspectos de la vida, tan importantes como la [salud pública](https://saludsinbulos.com/vacunas/noticias-falsas-vacunas-eficacia-campanas/).

Cambiando un poco de tema, esta sobreexposición a información en algunas ocasiones es contraproducente, ya que cuando quieres saber algo al instante pueden ocurrir dos cosas principalmente, que no sepas donde buscarlo o que no te fíes de todos los sitios donde se refleje eso que buscas.

Conociendo esta serie de problemas he decidido enfocar el proyecto en esta dirección, en ofrecer información de la manera más rigurosa y completa posible, eso sí, centrándome en resultados deportivos, y más concretamente en resultados de Fórmula 1, algo que quizás no es tan polémico como los ejemplos anteriores pero que a fin de cuentas es bastante información que debe ser organizada y mostrada coherentemente a un público, que quizás en España un poco menos, pero que va constantemente en auge.

Aparte del propósito anterior, la finalidad del proyecto es exponer todos los datos que haya disponibles relacionados con la Fórmula 1, de forma clara, concisa y directa y lo mejor de todo, de forma automatizada y al momento. Para ello nos basaremos en distintas fuentes de código abierto y gratuitas, creando una herramienta fácil de usar y sin ningún coste.

Antes de iniciar el proyecto, y para la realización de este se han tenido que investigar una serie de tecnologías. El resultado de este estudio se ha plasmado en los siguientes apartados.

**API REST**

Buscando una definición sencilla, REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad pero también mucha complejidad. A veces es preferible una solución más sencilla de manipulación de datos como REST.

REST cambió por completo la ingeniería de software a partir del 2000. Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por [Roy Fielding](https://roy.gbiv.com/), el padre de la especificación HTTP y uno los referentes internacionales en todo lo relacionado con la Arquitectura de Redes, en su estudio [‘Architectural Styles and the Design of Network-based Software Architectures’](https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm). En el campo de las APIs, REST (Representational State Transfer- Transferencia de Estado Representacional) es, hoy en día, el alfa y omega del desarrollo de servicios de aplicaciones.

Características de REST

* **Protocolo cliente/servidor sin estado:** cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché.
* Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
* Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado.
* Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
* **Sistema de capas:** arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
* **Uso de “hipermedios”:** “hipermedia” es un término acuñado por [Ted Nelson](http://hyperland.com/) en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Ejemplo de respuesta de una petición a una API REST según el principio HATEOAS:

Ventajas que ofrece REST para el desarrollo

* **Separación entre el cliente y el servidor:** el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas (MUY importante en nuestro proyecto), aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
* **Visibilidad, fiabilidad y escalabilidad:** La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el front y el back y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
* **La API REST siempre es independiente del tipo de plataformas o lenguajes:** la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

**FRONTEND Y BACKEND**

Según los últimos estudios cada uno de nosotros pasa más de 4 horas al día frente a un ordenador, durante las cuales interactuamos con un numero “X” de páginas web. No podemos negar que la web forma parte de nuestro día a día, aunque va más allá de los diseños agradables y animaciones, pues por detrás encontramos una serie de tecnologías que hacen que la magia sea posible. Estas tecnologías, se encargan de que todo funcione correctamente para que el usuario no tenga ningún problema al mandar una petición, consulta u obtener datos. Estamos hablando de los lados frontend y backend que se unen para cumplir un mismo objetivo: buscar la satisfacción del usuario.

El **Frontend** se enfoca en el usuario, en todo lo que podemos interactuar y lo que vemos mientras navegamos. Así, nuestra web busca causar una buena impresión y agradar al usuario, para lo cual utiliza HTML, CSS y JAVASCRIPT. Buena experiencia de usuario, inmersión y usabilidad, son algunos de los objetivos que busca un buen frontend y hoy en día existen una gran variedad de frameworks, preprocesadores y librerías que nos ayudarán en esta tarea. Para un frontend la creatividad es el recurso más valioso, ya que tendrá que tomar fuentes, colores, imágenes y todos lo recursos de los cuales disponga para crear sitios agradables que se vean bien en todos los dispositivos y resoluciones.

Por otro lado, tenemos el **Backend** enfocado en hacer que todo lo que está detrás de un sitio web funcione correctamente. Toma los datos, los procesa y los envía al usuario, además de encargarse de las consultas o peticiones a la Base de Datos, la conexión con el servidor, entre otras tareas que debe realizar en su día a día. Cuenta con una serie de lenguajes y herramientas que le ayudan a cumplir con su trabajo como PHP, Ruby, Python, JavaScript, SQL, MongoDb, MySQL, etc, estos son usados para crear sitios dinámicos. Como en muchos sitios la información se encuentra en constante cambio o actualización, una buena capacidad de respuesta y una velocidad óptima del sitio son responsabilidades que un backend debe de afrontar.

Ya que entendimos las funciones de cada uno, podemos afirmar que ninguno de los dos lados es mejor o menos importante que el otro, pues se complementan con el fin de cumplir el mismo objetivo…

**ANGULAR**

Es la herramienta que utilizaremos principalmente en nuestro proyecto, así que nos centraremos de lleno en ella en este apartado.

**Angular** es un **framework** de desarrollo para JavaScript creado por Google. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones web SPA y además darnos herramientas para trabajar con los elementos de una web de una manera más sencilla y optima.

Una aplicación web **SPA** creada con Angular es una web de una sola página, en la cual la navegación entre secciones y páginas de la aplicación, así como la carga de datos, se realiza de manera dinámica, casi instantánea, asincronamente haciendo llamadas al servidor (backend con un API REST) y sobre todo sin refrescar la página en ningún momento.

Antes de la aparición de este framework este tipo de aplicaciones web de una sola página se hacían usando solo JavaScript y jQuery, lo que era muy complejo, tedioso y difícil de mantener. Aparte de que necesitaba gran cantidad de código para hacer cualquier cosa. Este es el problema que viene a resolver Angular, convirtiendo el desarrollo de una web SPA en un proceso muy sencillo, muy limpio, más orientado a objetos y con una arquitectura basada en componentes y buenas prácticas.

Otra ventaja que tiene este framework es que está respaldado por Google y tiene una comunidad brutalmente grande detrás.

Ventajas de usar angular

1. **Facilidad de uso**

Dado que Angular es un framework, ofrece muchas funcionalidades de serie que una simple biblioteca. Con otro software similar, lo más común es tener que usar varias bibliotecas de terceros a la hora de desarrollar una aplicación. Lo más probable es que se necesite algunas adicionales para hacer el routing (enrutado de la aplicación web), para la gestión de dependencias, para realizar llamadas a APIs REST, para hacer testing, etc…

Con Angular ya sabes desde el primer momento cómo organizar el código, como se realizan las diferentes tareas que necesitas, la arquitectura de la aplicación…

1. **TypeScript**

Aunque se puede programar con ECMAScript (estándar de JavaScript) “puro”, el equipo de Angular decidió que haría todo el desarrollo con el lenguaje TypeScript, y casi toda la documentación utilizan este lenguaje.

Una de las ventajas de usar TypeScript en vez de JavaScript es la consistencia en la documentación. Por ejemplo, ES6 (o sea, ECMAScript 2015) ofrece varias formas diferentes de declarar un objecto, lo cual puede llegar a confundir. Con TypeScript esto no pasa, y toda la sintaxis y la manera de hacer las cosas en el código es la misma, lo que añade coherencia a la información y a la forma de leer el código.

Aunque Angular no te obliga a usar TypeScript, el equipo del core de Angular sí que lo ha adoptado y en la documentación sugiere usar TypeScript por defecto. Esta consistencia debería ayudar a evitar la confusión y la sobrecarga en la toma de decisiones derivadas de empezar con Angular.

Todo esto conlleva a largo plazo, en un mejor y más fácil mantenimiento de las aplicaciones.

1. **Componentes web**

Un componente en Angular es una porción de código que es posible reutilizar en otros proyectos de Angular sin apenas esfuerzo, lo que permite un desarrollo de aplicaciones mucho más ágil, pasando de un costoso MVC a una especie de puzle con los componentes.

El diseño de Angular adopta el estándar de los [componentes web](https://w3c.github.io/webcomponents/spec/custom/). Se trata de un conjunto de APIs que te permiten crear nuevas etiquetas HTML personalizadas, reutilizables y auto-contenidas, que luego puedes utilizar en otras páginas y aplicaciones web. Estos componentes personalizados en navegadores modernos y con cualquier biblioteca o framework de JavaScript que funcione con HTML.

El [soporte actual](https://www.webcomponents.org/) de este estándar por parte de los navegadores es muy reducido (básicamente Chrome), aunque existen [polyfills](https://www.imaginanet.com/blog/polyfills-para-css3-y-html5.html) para suplir esta carencia.