

universidad  
de león



**SIBI**

Sistema de recomendación de Hardware

Alejandro Abad Peláez.

4º Ingeniería Informática.

01/12/2020

# Resumen:

En esta memoria se detalla el móvil que ha llevado a la creación de este programa, la explicación de las herramientas con las que se ha llevado a cabo, una descripción del funcionamiento de la propia aplicación, así como del algoritmo que opera en el núcleo de la misma, un análisis de resultados, un análisis DAFO del proyecto, las posibles líneas de futuro del mismo y las conclusiones extraídas tras todo el trabajo realizado.

# Índice:

|  |    |
|--|----|
| 1. Resumen .....                             | 1  |
| 2. Glosario de Símbolos y Terminología ..... | 3  |
| 3. Introducción .....                        | 4  |
| 4. Desarrollo .....                          | 5  |
| 5. Conclusiones .....                        | 12 |
| 6. Anexos .....                              | 13 |
| 7. Bibliografía o referencias .....          | 14 |

# Glosario de Símbolos y Terminología:

Hardware

Periféricos

Oracle

Gaming

Input

Output

v-card

VOD

Raw

Streaming:

Build:

Carousel:

Footer:

Placeholder:

Copyright:

Cuello de botella

Feedback

Frontend

Backend

array

# Introducción:

Las bases de datos tradicionales, conformadas por tablas y relaciones llevan siguiendo un mismo modelo desde casi el momento en el que se inventaron. Empresas como Oracle llevan años y años ofreciendo exactamente lo mismo y hasta hace relativamente poco no parecía que esto fuese a cambiar. Sin embargo, con la llegada de la inteligencia artificial y la masiva cantidad de datos que a día de hoy se generan, debido a todos los dispositivos electrónicos conectados a Internet y la necesidad de un análisis rápido y sencillo de los mismos, nos han obligado a evolucionar en ciertos aspectos. Las bases de datos de grafos de conocimiento son una alternativa con cada vez mas oportunidades en el mercado de datos.

Su arquitectura permite un mejor manejo de la información, así como búsquedas más rápidas (al manejar nodos y diagramas de grafos, se pueden llevar a cabo búsquedas basadas en algoritmos de grafos, como el camino más corto), con relaciones inteligentes, lo que nos da acceso a hacer un uso de ellas semejante a la inteligencia artificial. Muchas páginas famosas, como Netflix o Amazon utilizan este tipo de bases de datos para ofrecer una experiencia personalizada a sus usuarios, encontrando patrones de forma sencilla en los gustos de usuarios en su base de datos, basándose en sus acciones y estadísticas dentro de la plataforma.

Si bien estas bases de datos cuentan con infinitas posibilidades, en lo que más parecen despuntar es a la hora de hacer sistemas de recomendación.

Aquí se presenta un sistema de recomendación de Hardware, pensado para que, tanto personas sin demasiados conocimientos de informática, como expertos en la materia que quieran facilitarse un poco la vida, puedan saber qué ordenador comprar para las tareas que necesiten o quieran llevar a cabo con él.

A continuación se detallará de forma técnica cómo funciona esta aplicación, cómo hace uso de las bases de datos de grafos de conocimiento y cómo pueden explotarse sus posibles aplicaciones más allá de un simple proyecto de universidad.

# Desarrollo:

## 4.1 El Problema

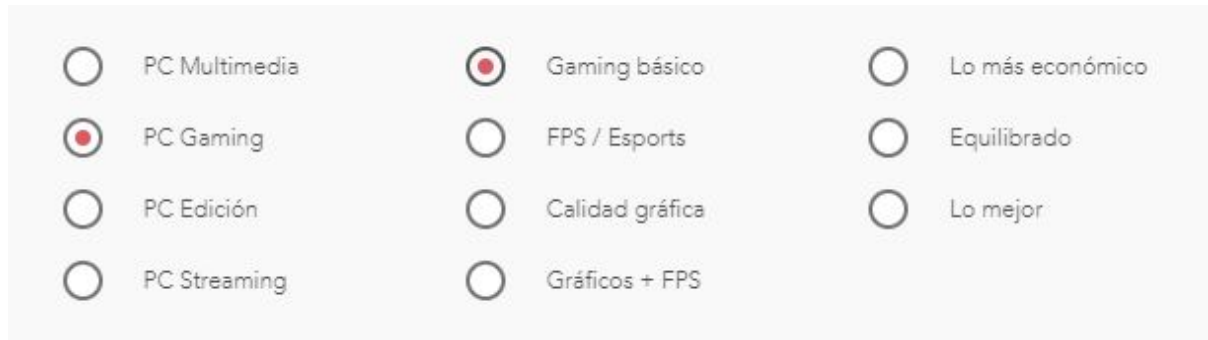
Hoy en día la informática está presente en prácticamente todos los ámbitos de nuestra vida diaria. El uso de computadoras o dispositivos con capacidad de procesamiento de datos está completamente extendido y adaptado a nuestra forma de vida y a la sociedad actual. Casi todo el mundo cuenta con al menos un ordenador personal en su casa, para el uso individual o compartido, ya sea pensando en el disfrute personal, en el entretenimiento o como una herramienta de trabajo.

La tecnología avanza a un ritmo vertiginoso y escoger un ordenador válido para lo que uno quiere desempeñar, puede convertirse en una tarea difícil y tediosa para la gran mayoría de la gente. Investigar e informarse acerca del tema es algo que, aunque por suerte, casi todos tenemos al alcance de nuestra mano, consume tiempo que muchas veces no tendremos o preferiremos dedicar a otras cosas. No todo el mundo tiene la capacidad para introducirse en este mundo y aprender desde 0. Esto pasa en todas las materias y es un problema del que todos somos conscientes.

Por ello, muchas empresas de venta de Hardware ponen a disposición de sus compradores, sistemas de filtrado que pueden hacer más rápidas y fáciles sus búsquedas, como el filtrado por marca, precio, modelo, etc. Sin embargo, para una persona que carezca de los conocimientos técnicos suficientes, estos filtros pueden no bastar para lograr encontrar y elegir lo que necesita. Incluso aquellos que tienen los conocimientos para hacer un filtrado eficiente, pueden verse perdidos en el mar de números, letras y nombres extraños que conforman los componentes de Hardware. Y, como ya hemos dicho antes, la tecnología avanza rápidamente, mantenerse al día no es tarea fácil, no siempre somos capaces de hacerlo.

Frente a este problema, tener un sistema de recomendación, que puede ser implementado en páginas de venta de hardware y tecnología en general, que se actualice con regularidad y que sea intuitivo y fácil de usar para aquellos que no tengan mucha idea, pero a la vez completo y con un nivel de abstracción suficientemente bajo como para que gente con conocimiento del tema pueda usarlo también cómodamente, puede ofrecer una solución perfectamente válida. Al hacer este tipo de aplicaciones caemos muchas veces en el error de abstraernos demasiado para hacerlo accesible a toda clase de público y nos olvidamos de que,

gente con más nivel de conocimientos puede no entender qué está eligiendo a bajo nivel, si es que le preocupa poder seleccionar con más detalle y precisión cada componente.



(Captura extraída de la web: <https://www.nategentile.com>)

En el ejemplo anterior nos encontramos con una elección que puede resultar demasiado transparente para un usuario con conocimientos. ¿Qué es exactamente “gaming básico”? ¿Qué criterio se está siguiendo para determinar lo que es básico? Es posible que para una persona, lo básico, sea jugar en calidad media, 30 fotogramas por segundo, mientras que para alguien con unos estándares más altos, “básico” sea jugar en la mejor calidad a 60 fotogramas.

Aparentemente, aquí nos encontramos ante una dicotomía, pues parece complicado suplir las necesidades de ambos perfiles a la vez. No obstante hay soluciones elegantes que pueden ser entendidas por todo el mundo y que más adelante en este documento se presentarán.

Y por esto decidí crear HWPicker (nombre susceptible a cambios en un futuro); como una solución para aquella gente que necesita comprar un ordenador nuevo, pero quizá no sabe demasiado de hardware o no puede dedicar mucho tiempo a informarse sobre qué es lo mejor para lo que necesita hacer. También es un sistema válido para los expertos en la materia. Al fin y al cabo, puede haber detalles de compatibilidad entre componentes o sinergias que igual pasamos por alto en un primer momento o combinaciones que resulten más eficientes que otras en contra de lo que la lógica nos dictaría. Incluso tan solo como una forma de tener una orientación para encaminar una posible decisión final y seguir investigando por tu cuenta, si llevas tiempo sin ponerte al día sobre las novedades en el mundo del hardware.

La aplicación, en una primera instancia, ha sido pensada como un sistema de recomendación pensado para ser implementado en la página de venta de Hardware PcComponentes, ya que me pareció más razonable y sencillo centrarme en una empresa que, si bien no es pequeña, nació siéndolo y es cercana a la gente, por lo

que presentarles el modelo y que se interesen por ello, resulta más fácil de imaginar que en el caso de presentarlo a empresas como Amazon u otras similares.

PcComponentes, como se mencionó antes de forma general, cuenta con un sistema de filtrado de hardware con opciones como filtrar por precio, por marca, por modelo, etc. Si bien estos filtros resultan sumamente útiles y cómodos para gente con cierta experiencia en la materia, para usuarios sin conocimientos, los resultados obtenidos tras aplicar los filtros pueden resultar ciertamente abrumadores.

Teniendo en cuenta, además, que PcComponentes cuenta con una herramienta llamada “configurador de Pcs”, que te permite seleccionar todos los componentes uno a uno de un ordenador para montártelo a tu gusto, un sistema de recomendación como el que ofrezco podría resultar muy beneficioso y podría agilizar y aumentar el número de ventas, ya que convertiría la herramienta en algo accesible para todo el mundo.



Aunque pueda parecer que no mucha gente sin conocimientos querría configurar un Pc a piezas, sobretodo en el mundo del gaming, la llamada “PC Master Race” (broma) ha adquirido mucha popularidad y se ha puesto de “moda”, a favor de una mejor experiencia. Por esa razón también, he decidido hacer especial hincapié en crear un input para gaming intuitivo y fácil de usar, pero completo para que los jugadores más exigentes puedan seleccionar al dedillo las características que su futuro ordenador ha de cumplir.

Centrarme en la base de datos de PcComponentes, además, me permite tener un acceso más directo a la información, así como a links de compra e imágenes fijas.

El propio algoritmo sería, realmente, lo que interesaría a PcComponentes. No obstante, una interfaz limpia, intuitiva y cuidada, así como una intencionalidad más allá de la propia recomendación, como es la parte de divulgación con la que cuenta la página, siendo casi una plataforma de aprendizaje, resulta una buena carta de presentación para que se genere un posible interés inmediato en ello.



## 4.2 Herramientas utilizadas

Las herramientas con las que hemos llevado a cabo el proyecto son diversas.



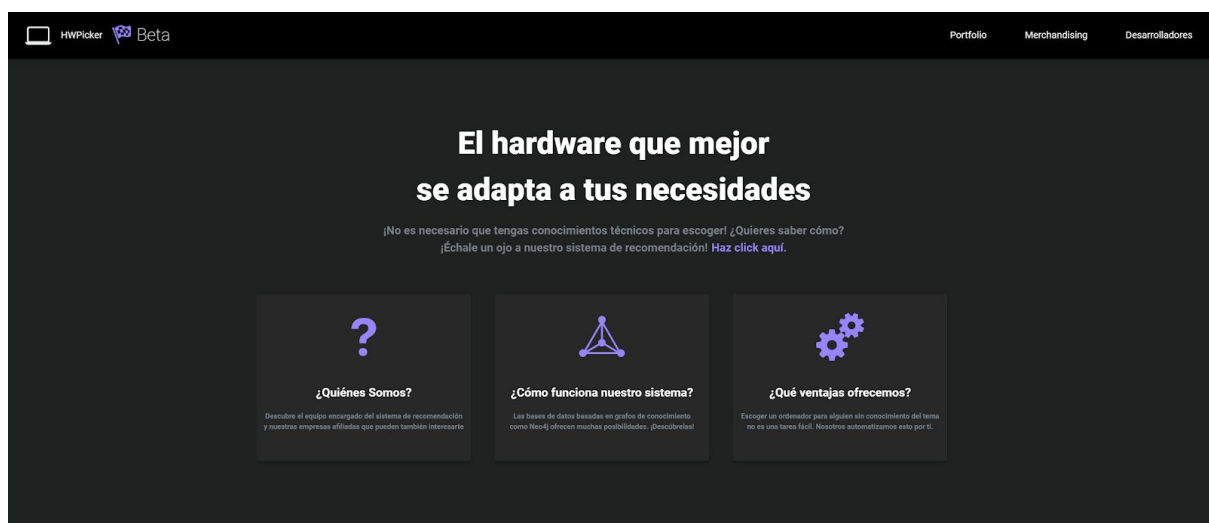
## 4.3 Funcionamiento de la aplicación

Procederé ahora a explicar, primero en un tono más informal, sirviéndome de la interfaz para ello y más adelante entrando en un nivel más profundo y técnico, la lógica interna del sistema de recomendación que he creado con capturas del código para explicar el algoritmo y la funcionalidad detrás de los elementos visuales de la página.

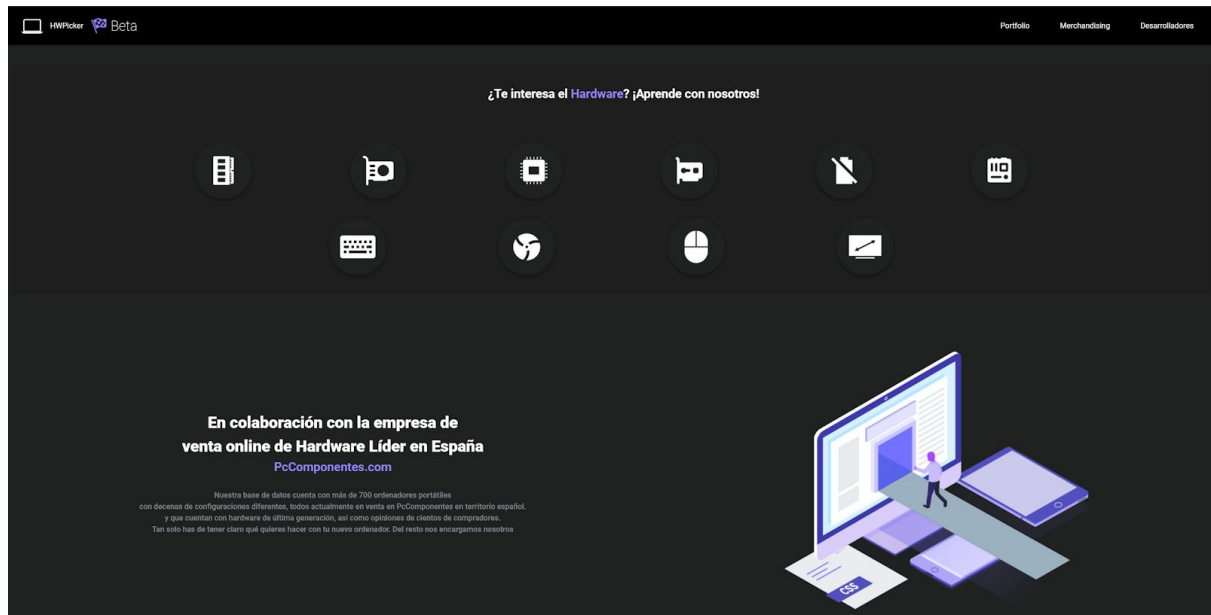
La interfaz de la aplicación se desarrolló pensando en que fuese, no solo un elemento para alojar el sistema de recomendación, sino también como una plataforma de aprendizaje y divulgación de hardware, para así complementar el propio sistema, de forma que incluso gente sin absolutamente ningún conocimiento pudiese aprovechar para entender, muy por encima, el funcionamiento básico de los componentes de un ordenador, así como todos los que son necesarios para que este funcione.

También quise que la propia página pudiese explicarse a sí misma en todos los aspectos, e intenté plantearla, más como un proyecto serio que como una práctica universitaria. No voy a entrar mucho en materia en este aspecto, ya que no es lo importante, por lo que simplemente explicaré un poco por encima cada una de las partes “extras” al objetivo del trabajo.

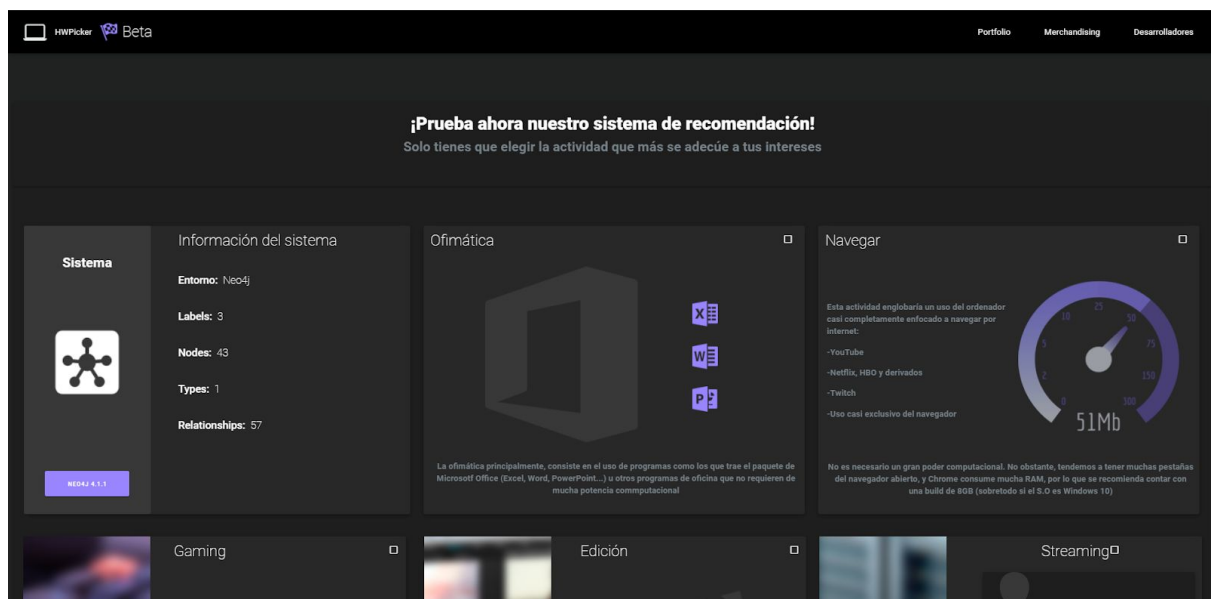
La parte superior de la interfaz cuenta con una barra de navegación en la que se halla el logo, el nombre de la página, una etiqueta indicando que está en fase de beta y más contenidos de interés. Justo debajo hay un “eslogan” publicitario, para atraer de primeras a los usuarios que entren, con algo de información sobre la página y sobre los desarrolladores de la misma, el sistema que se utiliza, la tecnología, etc y también una sección del texto destacada en la que se puede hacer click para que te lleve directamente al sistema de recomendación, ya que es lo importante.



Seguido a esto y en relación con lo que se comentó antes en cuanto al planteamiento de la página como una plataforma de aprendizaje, tenemos botones con iconos que hacen referencia a las distintas partes que conforman un ordenador, con una explicación sencilla y simple para entender qué papel desempeña cada uno y más abajo un poco de información sobre la propia página, para qué está pensado, así como la empresa con la que (se podría) colaborar.

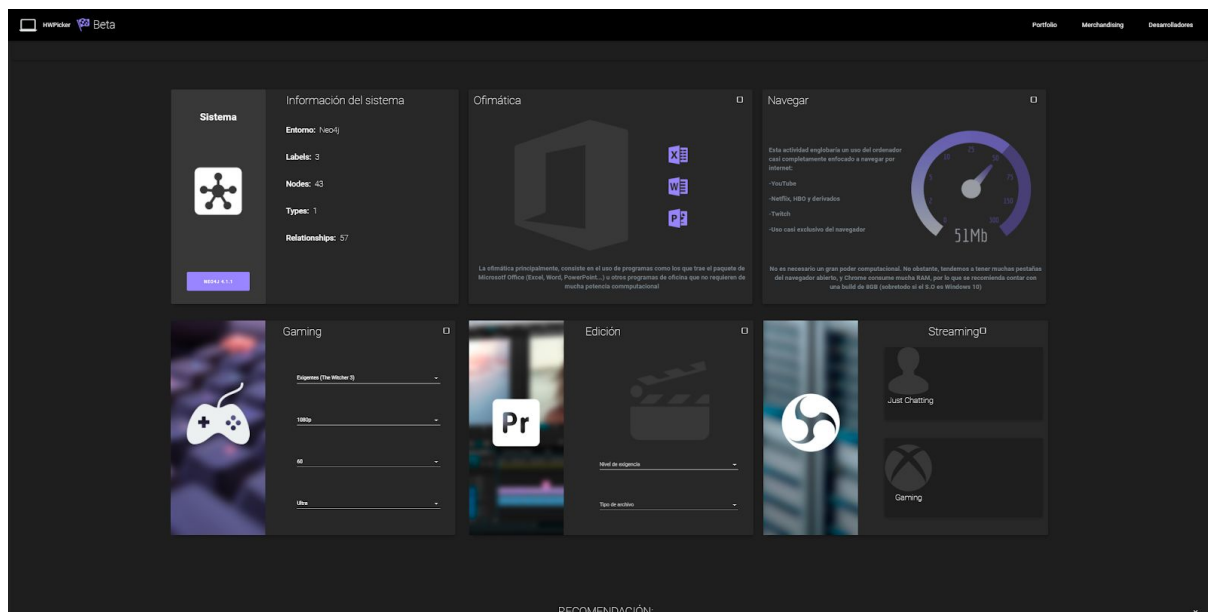


Después de estos extras con los que cuenta la aplicación tenemos ya la interfaz del propio sistema de recomendación como tal, el input desde el que recogeremos los datos con los que trabajará luego internamente el programa.



La interfaz del sistema de recomendación cuenta con 6 v-cards, 5 ellas correspondientes a las 5 actividades que he considerado más comunes a la hora de elegir un ordenador. La otra servirá para establecer ciertas características que nos ayudarán a ser más precisos en la recomendación que ofrecer, así como el botón de “recomiéndame”, que servirá para iniciar el procesamiento de los datos que nos devolverá las recomendaciones. Entre estas características se encuentra si queremos un PC pensando en el rendimiento puro, o si queremos una relación calidad-precio, si tenemos pensado hacer overclocking (esta opción es un “no” por defecto, ya que muchos usuarios no sabrán de qué se trata, aunque cuenta con una ventana de diálogo de ayuda a la que podremos acceder pulsando el icono colindante), así como el sistema operativo que pensamos usar y algo de información sobre la base de datos.

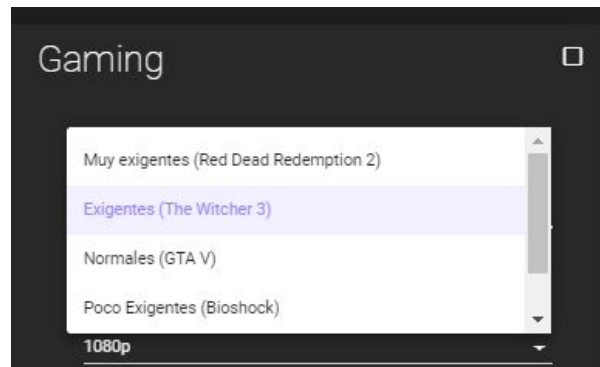
Las 5 actividades son: **Ofimática**, que vendría a ser trabajo de oficina y uso de programas como los incluidos en el paquete de microsoft office. **Multimedia** que sería un uso del ordenador pensado sobretudo para el consumo de VOD's, vídeos de youtube, películas, etc. Estas dos actividades son ciertamente simples, por lo que no cuentan con demasiadas opciones, más allá del sistema operativo que se utilizará, lo que determinará la cantidad de ram que se recomendará (esto ya lo explicaremos más adelante en profundidad), o, en el caso de multimedia, la resolución.



Las 3 actividades siguientes son más complejas y necesitan de más características que las anteriores para poder hacer una recomendación correcta de un ordenador para llevarlas a cabo.

**Gaming:** Ya que es la actividad en la que más experiencia tengo y una de las más complejas, le he dedicado mucho tiempo a hacerla todo lo completa y eficiente posible. Cuenta con 4 opciones que determinarán las características del ordenador que se recomendará. La primera opción es el tipo de juego, ya que hay juegos más exigentes y menos, mejor y peor optimizados... Esta primera opción abarcará 5

rangos, desde muy poco exigentes a muy exigentes, con un ejemplo de juego en cada una de ellas para que más o menos el usuario pueda hacerse una idea y pueda tener una referencia.



La siguiente opción es la resolución a la que se desea jugar que abarca las 3 principales: 1080p, 2K y 4K. Esta opción es muy importante a la hora de escoger un PC gaming, ya que procesar juegos en resoluciones altas puede resultar extremadamente exigente.

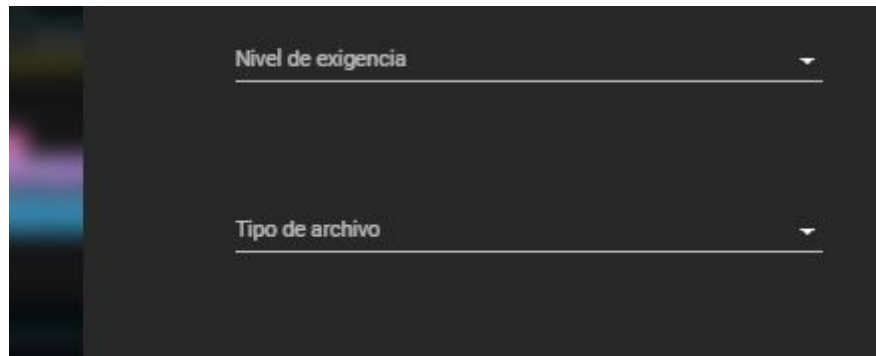
Luego contamos con la opción de fps (fotogramas por segundo), que, básicamente, determinará la fluidez con la que se ve el juego. De nuevo, una opción muy importante, pues alcanzar tasas altas de fps puede requerir de mucho poder de computación, dependiendo también de las dos opciones anteriores (no es lo mismo jugar a 120 fps un juego del 2008, que uno hecho ahora en 2020).

Por último tenemos la opción de la calidad a la que queremos jugar a los juegos, estando entre las posibilidades: calidad muy baja, baja, media, alta, muy alta y ultra.

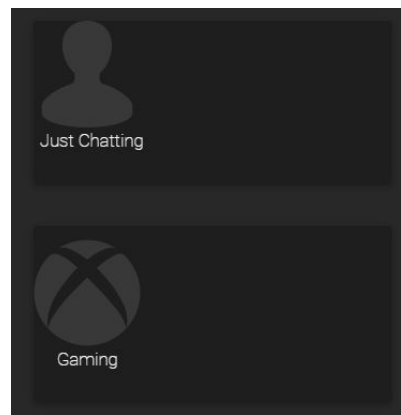
Más adelante veremos el resultado de combinar estas diferentes opciones y cómo se recomienda un ordenador en base a este, llegando a haber más de 300 combinaciones posibles.

Las dos actividades restantes son también complejas, pero al no tener tanto conocimiento acerca de ellas, no están tan completas como gaming.

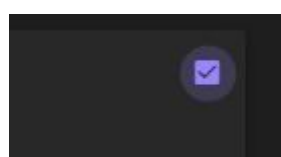
Lo más importante al recomendar un pc para edición, ya sea de vídeo o de fotografía, es tener en cuenta el tipo de archivo con el que se va a trabajar (no es lo mismo trabajar con un .mp4, que trabajar con archivos en bruto o "raw"), seguido de la calidad a la que se desea renderizar, ya que será mucho más exigente renderizar en 4K que en 1080 u otras resoluciones inferiores. De nuevo, se explicará más adelante la lógica interna que el programa tiene para recomendar ordenadores para este tipo de actividades.



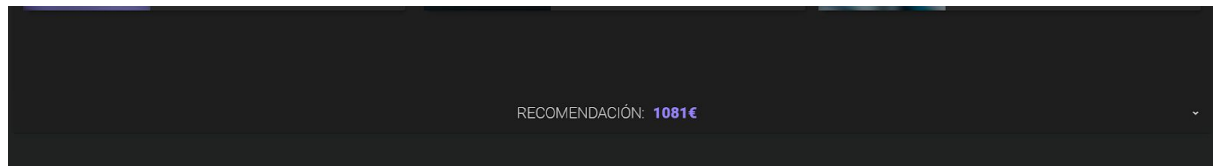
En cuanto a streaming, se ofrecen dos tipos de stream: lo que se llama just chatting, que es, simplemente el streamer hablando con los viewers, ya sea con la cámara puesta o sin ella con algún tipo de fondo o haciendo actividades que no requieran mayor poder de computación. Por otro lado tenemos gaming, que se trata de hablar, mientras la cámara te enfoca y jugar al mismo tiempo. Para este segundo tipo de streaming necesitaremos un ordenador potente, con mucha ram y mucho poder de procesamiento para que el streaming se vea fluido y la calidad del gameplay no se vea lastrada por el proceso de codificación de vídeo y el flujo de datos a través de la red.



Una vez hemos decidido las actividades que queremos llevar a cabo con nuestro ordenador, las marcaremos haciendo uso de la checkbox situada en la esquina superior derecha de cada una de ellas, pudiendo marcar una o más y procediendo a hacer click sobre el botón de la primera v-card: “recomiéndame”.

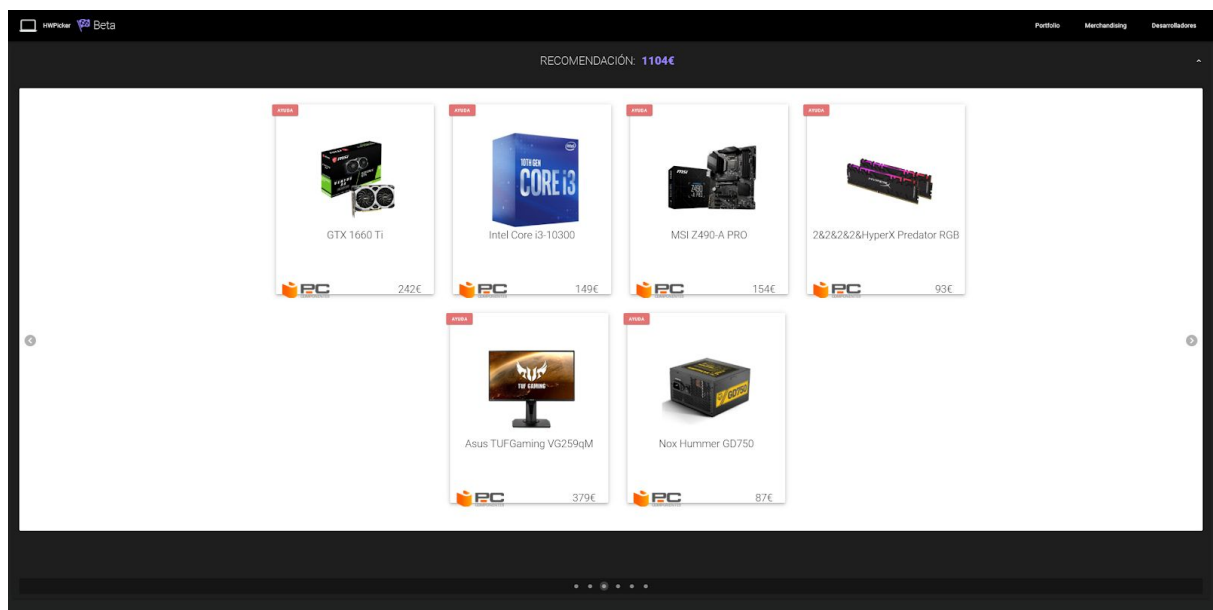


Una vez hayamos elegido las actividades y hayamos pulsado el botón de “recomiéndame”, los datos se enviarán al sistema, que hará los cálculos necesarios, que luego explicaremos en profundidad, y devolverá una o varias builds, que podremos usar para dichas actividades. Al pulsar en el botón, automáticamente, la aplicación nos hará un poco de scroll hacia abajo para que la vista encaje con la altura donde se sitúa la recomendación



La recomendación es un panel desplegable, que cuenta con un botón para hacerlo más user-friendly. Dentro del desplegable tendremos un elemento llamado carousel, que servirá para desplazar de izquierda a derecha o viceversa los contenidos que en él se encuentren. Al lado de la propia etiqueta de recomendación aparecerá el precio total de la build en euros, el cuál se actualizará si pasamos el contenido del carousel hacia un lado u otro.

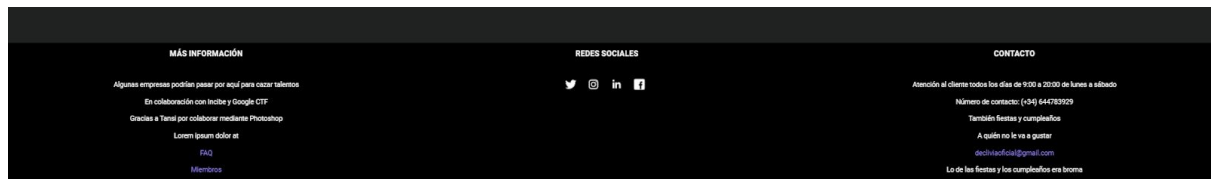
Si desplegamos la recomendación podremos ver todos los componentes que la build nos recomienda para la actividad o actividades que hayamos escogido. Cada componente estará situado en una v-card, en la que se mostrará una foto del mismo, su nombre, el precio que tiene en PcComponentes y un botón de ayuda en el que podremos hacer click para que se nos muestre una ventana de diálogo con una explicación breve del componente en sí.



Si hacemos click en cualquiera de los componentes, la aplicación nos redirigirá a PcComponentes, concretamente al link de compra de dicho componente.

Debajo del carousel podemos ver varios puntos, los cuáles serán las diferentes builds, entre las que podremos navegar pulsando en los mismos o en las flechas de los laterales del carousel.

Por último, en la parte inferior de la interfaz tendremos un footer con información de contacto, redes sociales, información de copyright y derechos y demás (en este caso la mayoría de cosas son placeholders)



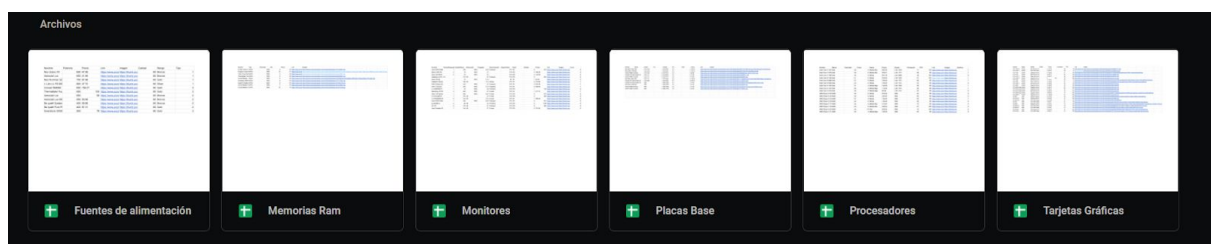
### 4.3.2 Base de datos

La base de datos de la aplicación la construí a mano, ya que los datos que necesitaba y el formato de los mismos era muy específico, por lo que no encontré ninguna preconstruida en internet que me sirviese.

Esta, está formada por los diferentes tipos de componentes que se recomiendan en mi aplicación, así como ciertas relaciones que existen entre algunos de ellos.

Para la construcción la base de datos, haciendo uso de excel, fui haciendo diferentes tablas para cada uno de los componentes, con sus respectivos atributos.

En la base de datos encontramos un total de 6 tablas, ya que consideré que lo mejor era recomendar los componentes “más importantes” de una build o los que, en general, más ayuda requieren para una correcta configuración.



Hay varios atributos que compartirán todas las tablas, como por ejemplo el nombre del componente, el link de compra en PcComponentes y la imagen del mismo.

Después, cada uno tendrá sus propios atributos específicos.

Una vez terminados los archivos excel con la información de cada componente, exporté los archivos en formato .csv, que coge los valores y los separa con comas en texto plano.



Estos archivos son posteriormente utilizados para crear la base de datos, importándolos con los siguientes comandos en Neo4j desktop. Si bien es cierto que se puede importar todo de una sola vez, si no tengo entendido mal, yo preferí importar uno a uno cada archivo, para ir controlando la aparición de errores. Esto es parte el script en cypher utilizado para ello. Ahora lo desgranaremos un poco:

```
LOAD CSV WITH HEADERS FROM 'http://192.168.2.21/Desktop/BaseDeDatos/procesadores.csv' AS line
WITH line
CREATE (p:procesador { nombre: line.Nombre, marca: line.Marca, Velocidad: toInteger(line.Velocidad), nucleos: toInteger(line.Cores),
precio: toInteger(line.Precio), socket: line.Socket, puntuacion: toInteger(line.Puntuación), tdp: toInteger(line.TDP), link:
line.Link, imagen: line.Imagen, graficos: toInteger(line.Graficos)})

LOAD CSV WITH HEADERS FROM 'http://192.168.2.21/Desktop/BaseDeDatos/placasBase.csv' AS line
WITH line
CREATE (pl:placaBase { nombre: line.Nombre, marca: line.Marca, socket: line.Socket, overclocking: toInteger(line.OC), chipset:
line.Chipset, Pci: line.PCI, ramSlots: toInteger(line.Slots), precio: toInteger(line.Precio), link: line.Link, imagen: line.Imagen})

LOAD CSV WITH HEADERS FROM 'http://192.168.2.21/Desktop/BaseDeDatos/tarjetasGraficas.csv' AS line
WITH line
CREATE (t:tarjetaGrafica { nombre: line.Nombre, marca: line.Marca, modelo: line.Modelo, vram: toInteger(line.VRAM), precio: toInteger(
line.Precio), puntuacion: toInteger(line.Puntuación), tdp: toInteger(line.TDP), link: line.Link, imagen: line.Imagen})

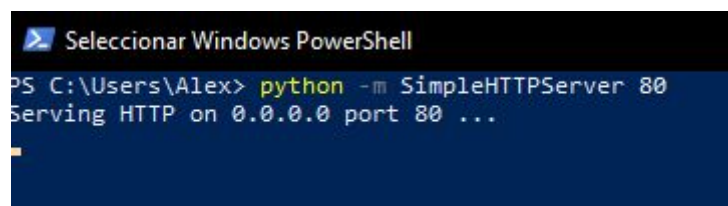
MATCH (p:procesador), (pl:placaBase)
WHERE p.socket = pl.socket
CREATE (p)-[:WORKSON]->(pl)

LOAD CSV WITH HEADERS FROM 'http://192.168.2.21/Desktop/BaseDeDatos/fuentesDeAlimentacion.csv' AS line
WITH line
CREATE (f:fuentesDeAlimentacion {nombre: line.Nombre, potencia: toInteger(line.Potencia), precio: toInteger(line.Precio), link:
line.Link, imagen: line.Imagen, calidad: toInteger(line.Calidad), rango: line.Rango, tipo: toInteger(line.Tipo)})

LOAD CSV WITH HEADERS FROM 'http://192.168.2.21/Desktop/BaseDeDatos/monitores.csv' AS line
WITH line
CREATE (m:monitor {nombre: line.Nombre, tiempoRespuesta: toInteger(line.TiempoRespuesta), tasaRefresco: toInteger(line.TasaRefresco),
resolucion: line.Resolución, pulgadas: toInteger(line.Pulgadas), sincronizacion: line.Sincronización, anguloVision: toInteger(
line.ÁnguloVisión), panel: line.Panel, bordes: toInteger(line.Bordes), precio: toInteger(line.Precio), link: line.Link, imagen:
line.Imagen, curvo: toInteger(line.Curvo)})
```

Como podemos ver se hace un load csv, indicando que se va a cargar un archivo con esta extensión, de una dirección concreta. Nótese que la dirección es una url, en lugar de ser un archivo. El procedimiento para conseguir esto es desplegar un pequeño servidor local con Python en el puerto 80 (el de http por defecto), de forma que Neo4j Desktop pueda encontrarlo, al ser una aplicación local de nuestro ordenador.

Para crear el servidor, en mi caso, abro una consola de windows (powerShell) y, situándome en el directorio por defecto cuando se abre, ejecutar el siguiente comando:



```
PS C:\Users\Alex> python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

Una vez ejecutado, se podrá acceder a todos los archivos situados en la ruta donde hemos ejecutado el comando como si fuese una URL, siempre y cuando la búsqueda se haga dentro de la propia red.

Para cada uno de los componentes, una vez cargado el archivo, se seguirá la instrucción CREATE, con la que podremos ir definiendo los atributos y su tipo (integer, string...) línea por línea.

Una vez creados los diferentes componentes, se establecí relaciones entre algunos de ellos.

Por ejemplo, una de las más importantes, las relaciones entre los procesadores y las placas base, que dependen del zócalo o socket. Si un procesador tiene un socket concreto, solo podrá ser utilizado en placas base con ese mismo socket.

```
MATCH (p:procesador), (pl:placaBase)
WHERE p.socket = pl.socket
CREATE (p)-[:WORKSON]->(pl)
```

La relación se da cuando el atributo socket del [label](#) procesador y el mismo atributo del label placaBase coinciden. La relación es de tipo "WORKSON", haciendo referencia a que dicho procesador funcionará sobre dicha placa base.

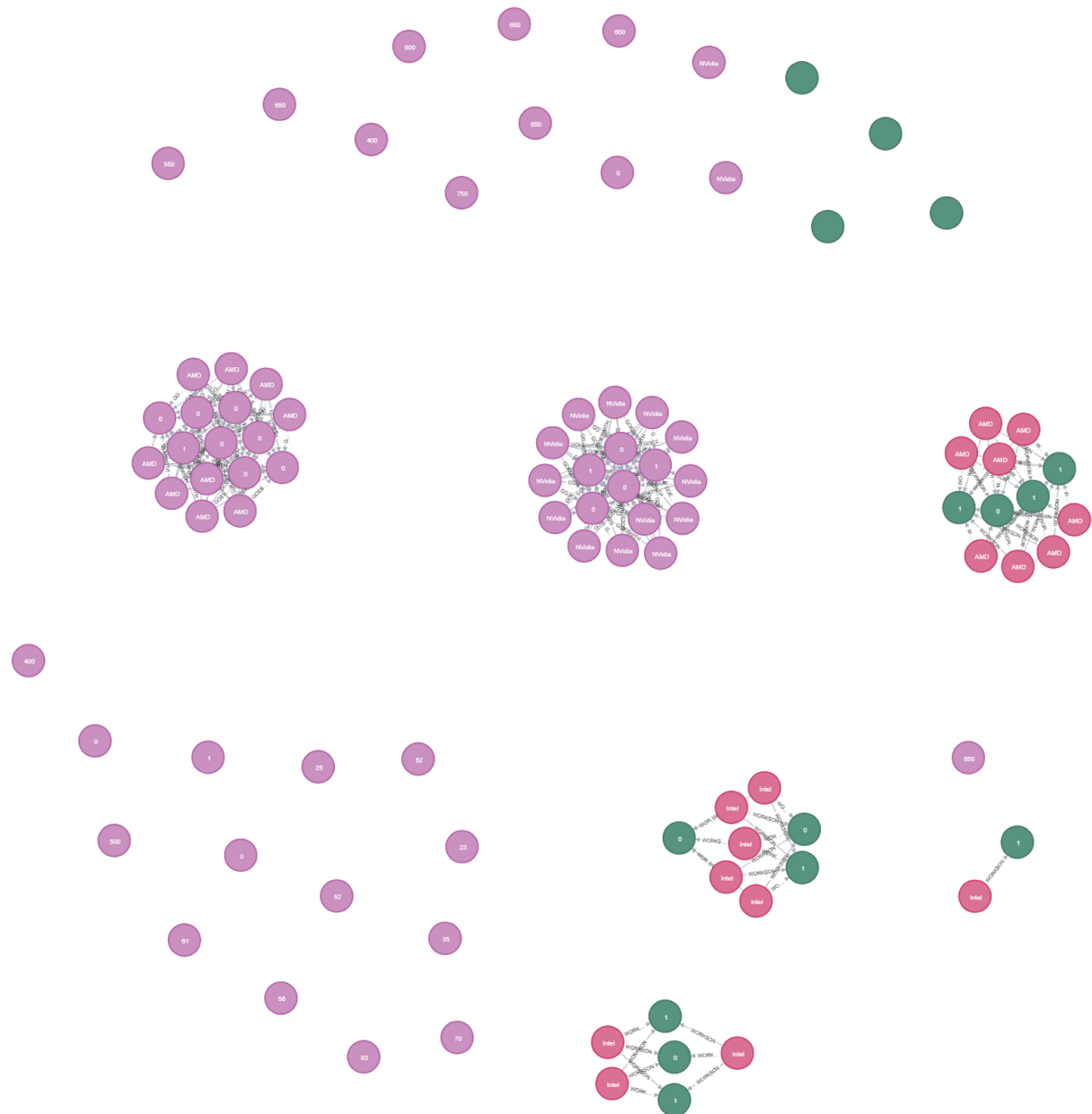
Otra relación importante es la de los monitores con las tarjetas gráficas. Los monitores tienen un atributo llamado "sincronización vertical", que sirve para que los fps que es capaz de conseguir la tarjeta gráfica y los hercios a los que funciona el monitor se sincronicen para evitar tirones y anomalías en las imágenes. Los monitores que cuentan con tecnología G-sync tendrán la posibilidad de sincronizarse con tarjetas gráficas de la marca NVidia, mientras que si el monitor cuenta con Freesync, se sincronizará con tarjetas de AMD.

```
MATCH (m:monitor), (t:tarjetaGrafica)
WHERE m.sincronizacion = "Freesync" AND t.marca = "AMD"
CREATE(t)-[:GOESWITH]->(m)

MATCH (m:monitor), (t:tarjetaGrafica)
WHERE m.sincronizacion = "G-sync" AND t.marca = "NVidia"
CREATE(t)-[:GOESWITH]->(m)
```

Esto no es absolutamente necesario, pero es recomendable tenerlo en cuenta a la hora de recomendar monitores.

## Foto de la base de datos



### 4.3.3 Funcionalidad, Backend y Algoritmo

Ahora procederé a explicar, de la forma más clara posible, la parte más técnica de la aplicación, tanto el funcionamiento de algunos de los componentes, como el propio algoritmo de recomendación.

Empezaremos con el código javascript situado en el frontend de la aplicación, que es el que se encarga de recoger los datos del input de la interfaz, procesarlos y enviárselos al backend para que trabaje con ellos el algoritmo de recomendación.

Dentro del script de nuestra vista principal, la que contiene toda la interfaz de la aplicación, ya que se trata de una single page application, encontraremos una variable llamada “actividades”, que será un array con los nombres de cada una de las 5 actividades disponibles dentro de nuestro sistema.

```
data: () =>({  
  //Las actividades del array irán en el siguiente orden: 0:jugar, 1:streaming, 2:edición, 3:navegar, 4:ofimática  
  actividades : {jugar: 0, streaming: 0, edición: 0, navegar: 0, ofimática: 0},
```

Este array nos servirá para que, cuando desde la interfaz, marquemos las checkbox de las actividades que queramos llevar a cabo con nuestro ordenador, podamos saber qué actividades han sido elegidas y qué rango de esfuerzo suponen de base.

```
<h1 class="myfontWhiteNormal mt-2">Streaming</h1>  
<v-spacer></v-spacer>  
<v-spacer></v-spacer>  
<v-checkbox  
  v-model="actividades.streaming"  
  class="mr-8"  
  value="8"  
  color="#9985FF"  
>  
</v-checkbox>
```

Se relaciona la checkbox con una de las actividades dentro del array mediante el atributo v-model que podemos ver en la imagen superior. En este caso, a la actividad streaming se le confiere un valor de 8 puntos, mientras que a ofimática, por ejemplo se le da un valor de 2 puntos en referencia a los requerimientos computacionales que tiene.

Esto lo hacemos para que el usuario pueda elegir con tranquilidad todas las actividades que quiera tener en mente. De forma transparente, el programa se encarga de, dentro las opciones escogidas, quedarse tan solo con las opciones más

exigentes, ya que si, por ejemplo, nos compramos un ordenador para gaming, vamos a poder hacer con él cosas menos exigente que gaming, como multimedia u ofimática. En breve mostraré y explicaré el código desarrollado para lograr esta transparencia.

Como vimos en la presentación de la interfaz, algunas actividades son lo suficientemente complejas como para necesitar características que las completen y sirvan para especificar más cuán potente ha de ser el ordenador elegido para desempeñarlas. Dentro de nuestro script contaremos también con arrays para las características que tienen actividades como jugar, o edición, así como otro array que guarde las características elegidas por el usuario.

```
//Creamos un vector de características e inicializamos cada una con un valor por defecto.  
caracteristicasJugar: {tipoDeJuego: "Exigentes (The Witcher 3)", resolucion: "1080p", fps: 60, calidad: "Ultra"},  
caracteristicasJuego: {tiposDeJuegos: ["Muy exigentes (Red Dead Redemption 2)", "Exigentes (The Witcher 3)", "Normales (GTA V)", "Pequeños (Angry Birds)"]}
```

En el caso de la actividad jugar, guardaremos en el array “caracteristicasJugar” la configuración elegida por el usuario, mientras que caracteristicasJuego tendrá todas las posibles opciones que el usuario podrá elegir.

Como podemos ver, caracteristicasJugar viene con unos valores por defecto, que sería lo que, en mi caso, considero un ordenador estándar. Lo que más se suele buscar a la hora de comprar un equipo para gaming.

Por otro lado, dentro de la propia v-card de gaming, contaremos con un v-select para poder elegir entre las opciones de cada una de las características.

```
<v-select  
  v-model="caracteristicasJugar.tipoDeJuego"  
  :items="caracteristicasJuego.tiposDeJuegos"  
  menu-props="auto"  
  label="Tipo De Juego"  
  hide-details  
  prepend-icon="mdi-map"  
  single-line  
  class="mr-15"  
></v-select>  
  
<v-select  
  v-model="caracteristicasJugar.resolucion"  
  :items="caracteristicasJuego.resoluciones"  
  menu-props="auto"  
  label="Resolución"  
  hide-details  
  prepend-icon="mdi-map"  
  single-line  
  class="mr-15"  
></v-select>
```

Mediante el atributo V-model, indicamos en qué posición del array se guardará la opción elegida por el usuario, mientras que con el atributo :items, indicamos de dónde ha de sacar las opciones disponibles, de entre los dos arrays anteriormente mencionados, respectivamente.

Una vez el usuario ha seleccionado todas las características de cada una de las actividades que ha seleccionado, pulsará en el botón “Recomiéndame”, situado en la primera v-card.

```
<v-btn height="60" width="200" class="primary" @click="recomendar()">  
  Recomendame  
</v-btn>
```

Como podemos ver, este botón llama a una función denominada “recomendar” cuando hacemos click sobre él. Esta función lleva a cabo varias cosas. Primero cambia el valor de 3 variables booleanas a false. Más adelante explicaremos su función. Después pone una de nuestras variables globales llamada precioBuild a 0, con la intención de “limpiar” el valor de dicha variable, y que no se mezcle con las recomendaciones de las builds siguientes.

```
//Vamos a recorrer las actividades en busca de cuáles han sido las marcadas y nos quedaremos solo con aquellas que tengan el valor más alto  
var mayor = 1;  
  
for (const actividad in this.actividades)  
{  
  if(this.actividades[actividad] > mayor)  
  {  
    mayor = this.actividades[actividad];  
  }  
}
```

Lo siguiente que hace es recorrer el array de actividades y se queda con el número más alto, el cuál hace referencia a la complejidad de la actividad. Una vez tenemos el número más alto, buscaremos todas las actividades con dicho valor y las guardaremos. De esta forma nos quedamos solo con las actividades de mayor complejidad, ya que un ordenador capaz de realizar estas, también podrá con actividades más simples, como ya explicamos antes y además nos permite ver si se han elegido actividades complementarias. Me explico: edición requiere un procesador potente, por lo que se prioriza este componente a la hora de recomendar un ordenador. Si se ha elegido edición y también ofimática, por ejemplo, como las dos actividades necesitan poder el procesador, con recomendar un ordenador para edición será suficiente, pues el procesador será lo suficientemente potente como para poder cumplir también con las tareas de ofimática, por lo que ofimática quedaría “descartada” y nos centraríamos solo en edición.

No obstante, si se elige gaming, donde se prioriza la tarjeta gráfica y se elige edición, donde se prioriza el procesador, la build resultante será una mezcla de procesador potente y gráfica potente, para poder llevar a cabo ambas actividades. A esto lo llamo actividades complementarias.

Buscamos, entonces, todas las actividades cuyo valor sea el mayor que hemos encontrado.



```

//Recorreremos una segunda vez el array para guardar las actividades marcadas cuya actividad computacional es mayor
var marcadas = [];
var counter = 0;

console.log("El número más grande es: " + mayor);

for(const actividad in this.actividades)
{
    if(mayor == this.actividades[actividad])
    {
        marcadas[counter] = actividad;
        counter++;
    }
}

```

En nuestra aplicación solo existen dos actividades complementarias; las ya mencionadas gaming y edición, por lo que, al buscar actividades con el mayor valor de complejidad, pondremos un contador al guardarlas. De esta forma, si el contador es mayor que 1, sabremos que el usuario ha escogido edición y gaming.

```

if(marcadas.length == 1)
{
    for(var act in marcadas)
    {
        switch (marcadas[act])
        {
            case "jugar":
                this.jugar();
                break;

            case "edicion":
                this.edicion();
                break;

            case "streaming":
                this.streaming();
                break;

            case "diseño":
                this.diseño();
                break;

            case "navegar":
                this.navegar();
                break;

            case "ofimatica":
                this.ofimatica();
                break;
        }
    }
}
else
{
    this.combinacion();
}
},

```

Si el valor del contador es 1, comprobaremos qué actividad es la elegida y llamaremos a la función encargada de procesar los datos de dicha actividad. Me centraré solo en la función de jugar, ya que las demás son bastante más simples.

Para jugar tenemos 4 características: tipo de juego, resolución, fps, calidad, cada una de ellas con varias opciones, lo que da un total de 300 combinaciones posibles. Mi intención es, mediante un algoritmo, obtener un resultado numérico en función de los valores escogidos que pueda representar fielmente la complejidad computacional de lo que queremos hacer. Este número tendrá estrecha relación con la puntuación dada a los componentes procesador y gráfica en la base de datos.

Dentro de la función jugar llamaremos, nada más entrar a otra función llamada “calcularEsfuerzoJugar”, con la que sacaremos dicho resultado.

```
calcularEsfuerzoJugar()
{
    //Si el tipo de juego es muy exigente utilizaremos un algoritmo concreto para calcular el esfuerzo
    var pruebaTipo = this.traducirTipoDeJuego();
    var pruebaRes = this.traducirResolucion();
    var pruebaFps = this.traducirFps();
    var pruebaCalidad = this.traducirCalidad();

    if(pruebaTipo == 50)
    {
        //Si la resolución es 4k le sumaremos 5 y los fps a partir de 60 se verán afectados
        if(pruebaRes == 15)
        {
            pruebaRes = pruebaRes + 5;

            if(pruebaFps >= 10)
            {
                pruebaFps = pruebaFps + 5;
            }
        }
    }

    //Una vez hemos modificado ciertos datos en caso
    var esfuerzo = pruebaTipo + pruebaRes + pruebaFps + pruebaCalidad;

    return esfuerzo;
},
```

A cada una de las características de jugar, les he asignado un número para poder trabajar con ellas. Primero sacaremos dicho número llamando a las diferentes funciones encargadas de traducir la característica a dicho valor. Estas funciones son extremadamente simples, reduciéndose tan solo a if else:

```
traducirTipoDeJuego()
{
    if(this.caracteristicasJugar.tipoDeJuego == "Muy exigentes (Red Dead Redemption 2)")
    {
        return 50;
    }
    else if(this.caracteristicasJugar.tipoDeJuego == "Exigentes (The Witcher 3)")
    {
        return 40;
    }
    else if(this.caracteristicasJugar.tipoDeJuego == "Normales (GTA V)")
    {
        return 30;
    }
    else if(this.caracteristicasJugar.tipoDeJuego == "Poco Exigentes (Bioshock)")
    {
        return 20;
    }
    else
    {
        return 10;
    }
},
```



El algoritmo para sacar el esfuerzo será tan solo una suma de los diferentes valores que pueden tomar las características. Con esa simpleza alcanzamos resultados muy buenos. Lo realmente complejo del algoritmo ha sido saber qué valores han de tomar las características y cómo se influyen unos a otros. El tipo de juego es algo así como el valor base, siendo múltiplos de 10. La resolución afecta a los fps, ya que si tenemos una resolución muy alta, alcanzar mayores tasas de fps es más complicado. Por último, tener un juego en calidad alta o baja puede variar en hasta 5 puntos.

Una vez tenemos todos estos valores, los sumamos y devolvemos el resultado a la función jugar.

Una vez tenemos el esfuerzo, crearemos una estructura de datos llamada data, donde meteremos información de la que tendremos que hacer uso en el backend. Por un lado las características que podemos marcar en la primera v-card, como el overclocking o calidad-precio (economic). A mayores guardaremos dos características de jugar que necesitaremos: la resolución y los fps. Con la estructura creada, haciendo uso de axios, que es una librería de JavaScript que nos permite hacer peticiones HTTP como clientes, le mandaremos esta información al backend. El backend procesa los datos y nos los devuelve, para que los mostremos por pantalla. En el siguiente apartado se explicará la función del backend, ahora vamos a centrarnos en terminar el funcionamiento del frontend.

```
async jugar()
{
  var esfuerzo = this.calcularEsfuerzoJugar();

  var overC = this.oc;
  var eco = this.economic;
  var resolucion = this.caracteristicasJugar.resolucion.split("p");
  var tasaRefresco = this.caracteristicasJugar.fps;

  var data = {
    puntuacion: esfuerzo,
    overclocking: overC,
    economic: eco,
    caract: {res: resolucion[0], tasaRef: tasaRefresco}
  }

  var respuesta = await axios.post(ip + '/priorizarGrafica', data)
  this.builds = respuesta.data;
}
```

La respuesta del backend la guardamos en una variable global llamada builds, y contendrá todas las recomendaciones hechas por el backend. Haremos ahora unas comprobaciones y lo siguiente será mostrar este contenido por pantalla.

Comprobaremos que la respuesta del backend no sea nula y en caso de serlo, mientras que el esfuerzo calculado sea mayor que 50, le iremos restando 5 puntos cada vez y volviendo a preguntar, hasta que nos devuelva algo. De esta forma, no habrá nunca recomendaciones vacías y podremos aproximarnos a un resultado

válido cuando lo que se nos pida sea demasiado potente y no tengamos nada que pueda hacer frente a esto.

```
while(this.builds.length == 0)
{
  if(data.puntuacion > 50)
  {
    this.esfuerzoDemasiadoAlto = true;
    console.log("No hemos obtenido ninguna build");
    data.puntuacion = data.puntuacion - 5;
    respuesta = await axios.post(ip + '/priorizarGrafica', data);
    this.builds = respuesta.data;
  }
}

if(data.puntuacion < 40)
{
  this.buildActivaLow = true;
  this.buildActiva = false;
}
else
{
  this.buildActiva = true;
  this.buildActivaLow = false;
}
```

Una vez hemos obtenido las builds y las hemos guardado en la variable global, nos encargaremos de mostrar el precio por pantalla y en caso de no haber encontrado nada lo suficientemente potente, un mensaje aclarándolo. Esto lo haremos cambiando el valor de variables booleanas, las mismas que en la función recomendar poníamos a false al principio, para que no saliese nada hasta no tener builds en una nueva recomendación.

Las recomendaciones se muestran en un carousel.

```
<v-carousel-item
  v-for="(build, i) in this.builds"
  :key="i"
>
  <v-card
    color="#ffffff"
    height="1400"
  >
    <v-col>
      <v-row justify="center" class="mt-10">
        <v-spacer></v-spacer>
        <v-card
          v-for="(item, i) in build.slice(0, 4)"
          :key="i"
          color="#ffffff"
          height="600"
          width="500"
          class="ml-15 hoverMouse prueba"
          elevation=6
          @click="redirect(item)"
        >
          <v-row>
            <div class="text-center">
              <v-dialog
                v-model="dialogCompleto[i]"
                max-width="1550"
              >
                <template v-slot:activator="{ on, attrs }">
                  <v-btn
                    color="red lighten-2"
                    dark
                    v-bind="attrs"
                    v-on="on"
                  >
```

En el carousel haremos un v-for que recorrerá toda nuestra variable global builds y para cada uno de los builds que tengamos, creará una v-card con la información pertinente, como su foto, modelo, precio y link de compra, al cuál nos redirigirá la página si hacemos click sobre la v-card.

```
</v-for>  
<h1 class="myfontRed mt-15" v-if="esfuerzoDemasiadoAlto">No hemos encontrado nada lo suficientemente potente en nuestro sistema, pero puede que esto se acerque</h1>  
</v-col>
```

También mostraremos esta frase en caso de que, de primeras, no se haya encontrado nada.

Vamos ahora con el backend de la aplicación.

El backend se ejecuta en nuestro ordenador en el puerto 3000, mediante Express, una librería de javascript que usa el protocolo HTTP, por lo que se puede hacer uso de todos los métodos de dicho protocolo.

Teniendo en cuenta los tipos de actividades, para optimizar el código y la aplicación, he dividido los algoritmos del backend en 3 principales. Uno se llama priorizarGrafica, otro priorizarProcesador y un último llamado priorizarAmbos. Las diferentes actividades dependerán de estas 3 funciones.

Todas ellas llamarán a otra función común a la que he denominado sacarComponentes, pasándole distintos parámetros y así obteniendo un resultado propio.

Como las 3 funciones son prácticamente iguales, y cambian solo en función de los parámetros que han de pasar y un par de cosas más, no muy grandes, procederé a explicar, por ejemplo, priorizarGrafica, matizando, cuando sea necesario, las diferencias que tiene con las otras dos funciones.

PriorizarGrafica es la función a la que llamábamos desde el frontend en la función de jugar. Más que una función como tal, es lo que se ejecuta cuando se lanza un mensaje post a la ip de mi ordenador, por el puerto 3000 indicando el directorio /priorizarGrafica. Mediante axios, desde el frontend podemos ejecutar el método post para llevar a cabo esto, como ya vimos en el código anteriormente referenciado

```
var respuesta = await axios.post(ip + '/priorizarGrafica', data)  
this.builds = respuesta.data;
```

Mediante el método axios.post, siendo la variable ip = <ip\_ordenador>:3000, podemos llamar a este trozo de código.

```

app.post("/priorizarGrafica", async function(req, res){

    var num = req.body.puntuacion;
    var oc = req.body.overclocking;
    var eco = req.body.economic;
    var caracteristicas = req.body.caract;
    var builds = [];

    //console.log(caracteristicas);

    console.log(num);

    await sacarComponentes(num, num + 5, num - 3, num, 0.70, 0.30, num, 1, oc, eco, caracteristicas, res);

});

```

Priorizar gráfica y el resto de funciones principales hacen, poco más, que desempaquetar los datos que le llegan por la variable req, para luego llamar a sacarComponentes, pasándole los atributos necesario para sacar un pc que pueda con la actividad y la potencia requerida.

La variable num, será el esfuerzo que le pasamos desde el frontend y representará la puntuación a la que se tienen que acercar los componentes que formarán nuestro pc. Ya que para gaming, que es el caso que estamos tratando, es más importante invertir más en un gráfica potente y dejar un poco más “de lado” al procesador, dentro de los parámetros de la función sacar componentes, indicamos que la puntuación de la gráfica que elijamos tendrá que ser, como mínimo, igual al esfuerzo que necesitamos suplir y pudiendo subir un máximo de 5 puntos más. Por otro lado, el procesador tendrá que tener una puntuación máxima del esfuerzo que le hemos pedido pudiendo subir 2 puntos más y una puntuación mínima de 3 puntos por debajo del esfuerzo. De esta forma conseguiremos builds que inviertan más en la gráfica que en el procesador y que resulten perfectas para gaming en relación calidad precio.

Los siguientes números que le pasamos a la función son el “peso” en cuanto a importancia que tienen los componentes para la build, teniendo la gráfica un 70% de relevancia, y el procesador un 30%. Más adelante veremos el algoritmo donde usaremos estos porcentajes. El resto de parámetros son características, como el overclocking, o el tipo de actividad en referencia numérica. Y por último tenemos la variable res, que nos permitirá devolver la builds a nuestro frontend.

Las otras dos funciones principales variarán los atributos, pasándole diferentes rangos de esfuerzo para gráfica y procesador, o cambiando los porcentajes de importancia de cada componente.

La siguiente función que nos interesa dentro del backend será sacarComponentes, la cuál se encargará de hacer las llamadas a nuestra base de datos, para sacar los componentes que necesitemos para cada caso.

```
//función para sacar los componentes de la build utilizando los rangos de puntuación para gráfica y procesador.
async function sacarComponentes(minGraf, maxGraf, minProc, maxProc, porcentajeGrafica, porcentajeProcesador, esfuerzo, flagActividad, overclocking, economico, características, res)
{
    var graficas = [];
    var procesadores = [];
    var modulosRam = [];
    var monitores = [];
    var fuentesAlimentacion = [];
    var builds = [];
    var queryGrafica = 'MATCH (g:tarjetaGrafica) WHERE g.puntuacion >= ' + minGraf + ' AND g.puntuacion <= ' + maxGraf + ' RETURN g';
    var queryProc = 'MATCH (p:procesador)-[:WORKSON]->(pl:placaBase) WHERE p.puntuacion >= ' + minProc + ' AND p.puntuacion <= ' + maxProc + ' RETURN p, pl'

    //Si el esfuerzo que hay que hacer es muy bajo, escogeremos un procesador algo más fuerte con APU, ya que no vamos a escoger tarjeta gráfica
    if(esfuerzo < 40)
    {
        minProc = 55;
        maxProc = 63;
        queryProc = 'MATCH (p:procesador)-[:WORKSON]->(pl:placaBase) WHERE (p.puntuacion >= ' + minProc + ' AND p.puntuacion <= ' + maxProc + ') AND p.graficos = 1 RETURN p, pl'

        if(flagActividad == 5)
        {
            minProc = 55;
            maxProc = 66;
            queryProc = 'MATCH (p:procesador)-[:WORKSON]->(pl:placaBase) WHERE (p.puntuacion >= ' + minProc + ' AND p.puntuacion <= ' + maxProc + ') AND p.graficos = 1 RETURN p, pl'
        }
    }

    //Si la actividad es poco exigente
    if(esfuerzo >= 40 && esfuerzo < 50)
    {
        minGraf = 50;
        maxGraf = 55;
        minProc = 50;
        maxProc = 55;
        queryGrafica = 'MATCH (g:tarjetaGrafica) WHERE g.puntuacion >= ' + minGraf + ' AND g.puntuacion <= ' + maxGraf + ' RETURN g';
        queryProc = 'MATCH (p:procesador)-[:WORKSON]->(pl:placaBase) WHERE p.puntuacion >= ' + minProc + ' AND p.puntuacion <= ' + maxProc + ' RETURN p, pl'
    }
}
```

Esta función empieza creando los arrays con los diferentes componentes que sacaremos de la base de datos, así como el array con todas la builds que generemos y las sentencias por defecto para sacar de la base de datos la tarjeta gráfica y el procesador.

Estas sentencias por defecto se centrarán en sacar ambos componentes atendiendo a las puntuaciones mínimas y máximas que hemos pasado por parámetro en las anteriores funciones principales.

En función de ciertas características estas sentencias de gráfica y procesador se verán alteradas. Por ejemplo, para actividades con una puntuación muy baja, como por ejemplo ofimática o multimedia, es mejor que el procesador sea un poco más potente, que tenga gráficos integrados y así no incluir una tarjeta gráfica dedicada, ya que no nos hará falta. De esta forma tenemos una build que se ajustará perfectamente a lo que queremos, ahorrándonos el dinero de una tarjeta gráfica.

Si la actividad es multimedia dedicaremos un poco más de presupuesto al procesador, subiendo de 63 puntos a 66, y conservaremos una build sin gráfica dedicada.

```
if(flagActividad == 5)
{
    minProc = 55;
    maxProc = 66;
    queryProc = 'MATCH (p:procesador)-[:WORKSON]->(pl:placaBase) WHERE (p.puntuacion >= ' + minProc + ' AND p.puntuacion <= ' + maxProc + ') AND p.graficos = 1 RETURN p, pl'
}
```

Por otra parte, si nos encontramos con una actividad como gaming que, dados los parámetros de entrada, el esfuerzo resultante sea muy bajo, como sigue siendo gaming, nos interesa tener una gráfica dedicada, por lo que seleccionaremos las más básicas.

Una vez tenemos preparadas las sentencias para la base de datos de gráfica y procesador, procederemos a llamar a Neo4j para que nos vaya devolviendo los resultados.

La sentencia del procesador nos devuelve tanto procesador como placa base.

```
queryProc = 'MATCH (p:procesador)-[:WORKSON]->(pl:placaBase) WHERE p.puntuacion >= ' + minProc + ' AND p.puntuacion <= ' + maxProc + ' RETURN p, pl'
```

Como un procesador puede tener compatibilidad con varias placas base, la base de datos nos devuelve varias veces el mismo procesador, cada vez con una placa base diferente. Me interesa tener un array en el que la primera posición sea el procesador, seguido de todas las placas bases con las que sea compatible. Cada uno de esos arrays los guardo dentro de otro array que inicializamos al principio de la función llamado procesadores. Las gráficas también las añadimos dentro de un array llamado graficas.

```
session
.run(queryGrafica)
.then(function(result){
    result.records.forEach(function(record){
        graficas.push(record._fields[0].properties);
    })
})
session
.run(queryProc)
.then(function(result){
    var procesadorActual = []; //Variable utilizada para marcar el procesador que se está devolviendo (pues este será devuelto tantas veces como placas base funcionen con él)
    result.records.forEach(function(record)
    {
        if(procesadorActual.length == 0) //Si es la primera iteración y la variable se encuentra vacía, la rellenamos
        {
            procesadorActual.push(record._fields[0].properties);
        }

        if(record._fields[0].properties.nombre == procesadorActual[0].nombre) //Mientras sea el mismo procesador guardaremos en un array sus placas base
        {
            procesadorActual.push(record._fields[1].properties)
        }
        else
        {
            procesadores.push(procesadorActual);
            procesadorActual = [];
            procesadorActual.push(record._fields[0].properties);
        }
    })
})
```

Ahora sacamos la RAM, y los monitores. Dependiendo de la actividad, el tipo de monitor que saquemos de la base de datos, tendrá unas características u otras. Los monitores gaming tendrán un panel IPS o VA, ya que tienen buena reproducción de color. Además de esto, tendrán que tener una resolución igual a la que quiera jugar el usuario. De nada sirve querer jugar a 4K si tienes un monitor 1080p.

Estas son las condiciones imprescindibles, a mayores tenemos una que intentaremos que se cumpla, pero de no ser posible, dará igual: la tasa de refresco. Si un jugador quiere jugar a 120 fps necesitará un monitor cuya tasa de refresco sea, como mínimo, 120 hercios. No obstante, como hemos puesto ya varias restricciones es posible que no se encuentre ninguno con estas características, pero la recomendación siempre ha de seguir adelante.



```

session
.run('MATCH (r:ram) RETURN r')
.then(function(result){
  result.records.forEach(function(record)
  {
    //console.log(record._fields)
    modulosRam.push(record._fields[0].properties);
  })
  //session.close();
  //Vamos a sacar ahora los monitores
  var queryMonitores = "";
  if(flagActividad == 1)
  {
    queryMonitores = "MATCH (m:monitor) WHERE (m.panel = 'VA' OR m.panel = 'IPS') AND m.resolucion = '' + caracteristicas.res + '' OPTIONAL MATCH(m) WHERE m.tasaRefresco > '' + caracteristicas.tasaRef + '' RETURN m";
  }
  //Si es ofimática, elegiremos monitores con panel TN
  else if(flagActividad == 4)
  {
    queryMonitores = "MATCH (m:monitor) WHERE m.panel = 'TN' AND m.resolucion = '' + caracteristicas.res + '' OPTIONAL MATCH(m) WHERE m.tasaRefresco > '' + caracteristicas.tasaRef + '' RETURN m";
  }
  else if(flagActividad == 5)
  {
    queryMonitores = "MATCH (m:monitor) WHERE m.panel = 'IPS' AND m.resolucion = '' + caracteristicas.res + '' OPTIONAL MATCH(m) WHERE m.tasaRefresco > '' + caracteristicas.tasaRef + '' RETURN m";
  }
  session
  .run(queryMonitores)
  .then(function(result){
    result.records.forEach(function(record)
    {

```

(PENDIENTE DE REVISIÓN)

Por otra parte, actividades como ofimática pueden realizarse con monitores con paneles TN ya que, a pesar de tener una peor reproducción de colores, tienden a ser mucho más baratos. Multimedia o edición requerirán de paneles IPS ya que interesa que la imagen se vea lo mejor posible.

Por último sacaremos las fuentes de alimentación, que para multimedia y ofimática no requerirán de mucha potencia, mientras que para las demás actividades, sí. Una vez guardados todos los resultados en sus respectivos arrays, se llama a la función calcularBuilds, a la que le pasaremos todos los arrays.

```

session
.run(queryMonitores)
.then(function(result){
  result.records.forEach(function(record)
  {
    //console.log(record._fields);
    monitores.push(record._fields[0].properties);
  })
  var queryFuentes = "";
  if(flagActividad == 1 || flagActividad == 5)
  {
    queryFuentes = "MATCH (f:fuentesAlimentacion) WHERE f.potencia >= 600 RETURN f";
  }
  else if(flagActividad == 4)
  {
    queryFuentes = "MATCH (f:fuentesAlimentacion) WHERE f.potencia <= 600 RETURN f";
  }
  session
  .run(queryFuentes)
  .then(function(result){
    result.records.forEach(function(record){
      fuentesAlimentacion.push(record._fields[0].properties);
    })
  })
  .catch(function(err){
    console.log(err);
  })
  .then(async function(){
    builds = await calcularBuild(graficas, procesadores, modulosRam, monitores, fuentesAlimentacion, porcentajeGrafica, porcentajeProcesador, esfuerzo, flagActividad, overlocking, economico);
    //pruebaRecibir(builds);
    res.send(builds);
  })
  .catch(function(err){
    console.log(err);
  })

```

Calcular builds se encargará de hacer diferentes combinaciones de componentes y calcular una puntuación para distinguir las que más potencial tienen.

Esta función, primeramente hará una distinción cuando el array de gráficas está vacío o cuando no lo está. Si el array de gráficas está vacío, significa que la build no constará de gráfica dedicada.

Si hay gráficas, el algoritmo recorrerá cada una de las gráficas y para cada una de ellas, recorrerá todos los procesadores, para sacar todas las combinaciones posibles.

Si no hay gráficas la cosa no cambia mucho, simplemente se analiza la build solo por el procesador.

Para cada combinación, ya tenga gráfica o no, se harán una serie de filtros y ajustes para que todos los componentes encajen. Primero se añadirán gráfica y procesador al array buildAux, que guardará todos los componentes de una build completa.

```
//Esta función probará combinaciones y calculará su puntuación para la actividad de la que se quiere hacer uso
async function calcularBuild(graficas, procesadores, modulosRam, monitores, fuentesAlimentacion, porcentajeGrafica, porcentajeProcesador, esfuerzo, flagActividad, overlocking, economico)
{
    builds = [];

    //Si no hay gráficas, significa que la build no consta de gráfica dedicada.
    if(graficas.length == 0)
    {
        for(proc in procesadores)
        {
            //Guardaremos la gráfica, el procesador y la puntuación de la build
            buildAux = [];
            var flagAniadir = true; //Este flag nos valdrá para saber si se ha añadido la build del procesador y la gráfica si llega a la puntuación requerida

            buildAux.push(procesadores[proc][0]);

            var posiblesPlacas = [];
            var placaElegida;
            var posiblesRam = [];
            var posiblesMonitores = [];
            var validPl = 0;
            var ram = 0;

            //Si la combinación de procesador y gráfica ha alcanzado una puntuación correcta, se procederá a calcular el resto de componentes
            if(flagAniadir)
            {
                //Ahora elegimos una placa base
                if(overlocking == 1)
                {
                    var counter = 0;

                    for(pl in procesadores[proc])
                    {
                        //Nos saltamos la posición 0, que será donde esté el propio procesador. Solo nos interesan las placas base
                        if(counter != 0)
                        {
                            if(pl.overlocking == 1)
                            {
                                posiblesPlacas.push(pl);
                                validPl++;
                            }
                        }
                        counter++;
                    }
                }
            }
        }
    }
}
```

Si el array de gráficas no está vacío, se probarán combinaciones y para cada una de ellas, se calculará una puntuación en base a lo eficientes que se consideran para la tarea que se requiere de ellas. Esto lo hacemos mediante el siguiente algoritmo, que hace la media de puntuaciones entre gráfica y procesador, teniendo en cuenta el porcentaje de peso de cada uno, los números de los que ya habíamos hablado antes, y de esas nos quedamos solo con aquellas que tienen una puntuación mayor o igual al esfuerzo necesario para la actividad.

```
puntuc = graficas[graf].puntuacion * porcentajeGrafica + procesadores[proc][0].puntuacion * porcentajeProcesador;

//Si la puntuación de la build alcanza o supera lo que necesitamos, lo guardaremos
if(puntuc >= esfuerzo)
{
    buildAux.push(graficas[graf]);
    buildAux.push(procesadores[proc][0]);
    flagAniadir = true;
}
```

Empezaremos filtrando las placas base, en función de si el usuario tiene pensado, o no, hacer overlocking y una vez pasados ese filtro, si el usuario ha escogido un ordenador calida-precio, se hará un segundo filtrado escogiendo la placa base más barata entre las disponibles. Si una placa base no cumple los requisitos, la combinación de procesador, gráfica y esa placa base, se descartará.

Si la build no es económica, elegiremos, de entre las placas válidas una al azar.



```

    }
  }
  else
  {
    validPl = -1;
  }

  //ValidPl solo valdrá 0 si no se ha encontrado ninguna placa para hacer overclocking para este procesador
  if(validPl != 0)
  {
    if(economico == 1 && validPl > 0)
    {
      var precioMin = 100000000;

      for(pl in posiblesPlacas)
      {
        //Escogemos la más barata
        if(pl.precio < precioMin)
        {
          precioMin = pl.precio;
          placaElegida = pl;
        }
      }
    }
  }
}

```

Una vez elegida la placa base se añadirá a un array que conformará los componentes de una build, el ya mencionado buildAux.

```

}
//Si la build no es económica, nos quedaremos con una placa base al azar
else if(economico == 0 && validPl > 0)
{
  var numrand = Math.round(Math.random() * (procesador.length - 1) + 1);

  placaElegida = posiblesPlacas[numrand];
}
else if(economico == 1 && validPl < 0)
{
  var precioMin = 100000000;

  for(pl in procesadores[proc])
  {
    //Escogemos la más barata
    if(pl.precio < precioMin)
    {
      precioMin = pl.precio;
      placaElegida = pl;
    }
  }
}
else if(economico == 0 && validPl < 0)
{
  var numrand = Math.round(Math.random() * (procesadores[proc].length - 1) + 1);

  placaElegida = procesadores[proc][numrand];
}

buildAux.push(placaElegida);

```

Lo siguiente será escoger la ram, que en función de la actividad y del esfuerzo de esta será una cantidad u otra

```

if(placaElegida != undefined)
{
    //Ahora escogeremos la ram

    //Si la actividad es jugar
    if(flagActividad == 1)
    {
        if(esfuerzo < 60)
        {
            ram = 8;
        }
        else
        {
            ram = 16
        }
    }

    //Si la actividad es edición
    else if(flagActividad == 2)
    {
        if(esfuerzo < 80)
        {
            ram = 16;
        }
        else if(esfuerzo >= 80 && esfuerzo < 110)
        {
            ram = 32;
        }
        else
        {
            ram = 64;
        }
    }
    else if(flagActividad == 4)
    {
        ram = 4;
    }
    else if(flagActividad == 5)
    {
        ram = 8;
    }
}

```

(PENDIENTE DE REVISIÓN)

Una vez escogida la cantidad de ram, calcularemos las diferentes combinaciones de módulos para la placa base elegida, en función de la cantidad de ranuras que esta tenga. Por ejemplo, si una build necesita 16 GB de RAM y tenemos una placa con 4 ranuras podremos poner 4 módulos de 4 GB, 2 módulos de 8GB o 1 módulo de 16GB.

```

//var capacidad = ram / placaElegida.ramSlots
var capacidadInicial = 4; //Empezamos con módulos de la capacidad más pequeña que tenemos en la base de datos
var capacidad = 4;
var modulos = 0;
var combinacionesRam = []
//Vamos a ver cuántos módulos de ram necesitamos con esa capacidad para suplir la cantidad que necesitamos total de RAM
while (capacidadInicial <= ram)
{
    while(capacidad <= ram)
    {
        capacidad = capacidad + capacidadInicial;
        modulos++;
    }
    var combinacionesRamAux = [];
    combinacionesRamAux.push(modulos, capacidadInicial);
    combinacionesRam.push(combinacionesRamAux);

    capacidadInicial = capacidadInicial * 2;
    capacidad = capacidadInicial;
    modulos = 0;
}

var posiblesRam = []

```

Si el usuario no ha escogido un pc económico, seleccionaremos los módulos de ram más rápidos. Si es económico seleccionaremos los más baratos.

```

//Si el pc no es económico, escogeremos los módulos RAM más rápidos.
if(economico == 0)
{
    //Vamos a ir iterando por el array en el que antes guardamos la cantidad de módulos y la capacidad de los mismo (por ejemplo 2x8GB o 4x4GB)
    for(combinacion in combinacionesRam)
    {
        var ramElegida;
        var velocidad = 0;
        var comb = 0;

        //Para cada una de las capacidades buscaremos el módulo de ram adecuado
        for(r in modulosRam)
        {
            //Primero buscamos que la capacidad en GB sea la misma
            if(modulosRam[r].gb == combinacionesRam[combinacion][1])
            {
                //Y luego buscamos el que tenga la mayor velocidad
                if(modulosRam[r].velocidad > velocidad)
                {
                    velocidad = modulosRam[r].velocidad;
                    ramElegida = modulosRam[r];
                    comb = combinacion;
                }
            }
        }

        var ramAux = [];

        ramAux.push(combinacionesRam[comb][0]);
        ramAux.push(ramElegida);

        //Guardaremos la el módulo elegido junto a cuántos de ellos necesitaremos para llegar a la ram requerida (1x16GB, 2x8Gb...)
        posiblesRam.push(ramAux);
    }
}

```

Guardaremos en un array el módulo escogido junto con la cantidad del mismo (por ejemplo 2 módulos de 8GB marca gigabyte...)

Código si escogemos los módulos más baratos.

```

else
{
    for(combinacion in combinacionesRam)
    {
        var ramElegida;
        var precio = 100000000000;
        var comb = 0;

        //Para cada una de las capacidades buscaremos el módulo de ram adecuado
        for(r in modulosRam)
        {
            //Primero buscamos que la capacidad en GB sea la misma
            if(modulosRam[r].gb == combinacionesRam[combinacion][1])
            {
                //Y luego buscamos el que tenga la mayor velocidad
                if(modulosRam[r].precio < precio)
                {
                    precio = modulosRam[r].precio;
                    ramElegida = modulosRam[r];
                    comb = combinacion;
                }
            }
        }

        var ramAux = [];

        ramAux.push(combinacionesRam[comb][0]);
        ramAux.push(ramElegida);

        //Guardaremos la el módulo elegido junto a cuántos de ellos necesitaremos para llegar a la ram requerida (1x16GB, 2x8Gb...)
        posiblesRam.push(ramAux);
    }
}

```

Ahora escogeremos un monitor para nuestra build. Si tenemos gráfica integrada haremos un primer filtrado, buscando monitores cuya tecnología de sincronización vertical sea compatible con la marca de la tarjeta gráfica, y también buscaremos los

que menor tasa de respuesta tengan. La tasa de respuesta es el tiempo que tardan en representar por pantalla lo que se hace en el ordenador.

```
if(flagActividad == 1)
{
    for(monitor in monitores)
    {
        if(graficas[graf].marca == 'AMD')
        {
            if(monitores[monitor].sincronizacion == "Freesync" || monitores[monitor].sincronizacion == "ambas")
            {
                posiblesMonitoresAux.push(monitores[monitor]);
            }
        }
        else
        {
            if(monitores[monitor].sincronizacion == "G-sync" || monitores[monitor].sincronizacion == "ambas")
            {
                posiblesMonitoresAux.push(monitores[monitor]);
            }
        }
    }

    var tiempoRespuesta = 100;

    for(monitor in posiblesMonitoresAux)
    {
        if(posiblesMonitoresAux[monitor].tiempoRespuesta < tiempoRespuesta)
        {
            tiempoRespuesta = posiblesMonitoresAux[monitor].tiempoRespuesta;
        }
    }

    for(monitor in posiblesMonitoresAux)
    {
        if(posiblesMonitoresAux[monitor].tiempoRespuesta == tiempoRespuesta)
        {
            posiblesMonitores.push(posiblesMonitoresAux[monitor]);
        }
    }
}
```

Por último, para cada uno de los arrays de componentes viables para la build, escogemos uno al azar, creamos nuestra build y devolvemos el array.

## 4.4 Análisis de resultados

Vamos a proceder ahora a analizar un poco los resultados que nos ofrece nuestro sistema de recomendación.

Empecemos viendo qué se nos recomienda cuando buscamos un Pc para ofimática.

Como podemos ver, una build baratita, con un procesador con gráficos integrados, sin tarjeta gráfica dedicada, 4 GB de RAM, una fuente de alimentación con certificación 80 plus Bronze y un monitor con panel TN.

RECOMENDACIÓN: 492€

| Componente             | Imagen | Nombre                         | Precio |
|------------------------|--------|--------------------------------|--------|
| Procesador             |        | Intel Core i3-10100            | 114€   |
| Placa base             |        | GIGABYTE B365M-DS3H            | 75€    |
| RAM                    |        | 1&1 Team Group Delta White RGB | 25€    |
| Monitor                |        | Acer KG251QJbmidpx             | 219€   |
| Fuente de alimentación |        | Be quiet! System Power 9       | 59€    |






Otra build, bastante parecida, con un monitor más barato, con una fuente de alimentación gold, con una placa base con algo más de calidad.

RECOMENDACIÓN: 490€

| Componente             | Imagen | Nombre                         | Precio |
|------------------------|--------|--------------------------------|--------|
| Procesador             |        | Intel Core i3-10300            | 149€   |
| Placa base             |        | MSI Z490-A PRO                 | 154€   |
| RAM                    |        | 1&1 Team Group Delta White RGB | 25€    |
| Monitor                |        | Asus VP228                     | 95€    |
| Fuente de alimentación |        | Be quiet! Pure Power 11        | 67€    |

Veamos qué se nos recomienda para multimedia.

RECOMENDACIÓN: 564€

|  |   |   |  |
|--|---|---|--|
| <br>Intel Core i5-9600K<br>189€ | <br>ASUS PRIME H310M-E R2.0<br>73€ | <br>1&1&HyperX Predator RGB<br>93€ | <br>BenQ GW270E<br>168€ |
| <br>Aerocool Lux<br>41€         |   |   |  |







En este caso contamos con un procesador algo más potente, también con gráficos integrados, 8GB de RAM, perfectos para poder tener varias pestañas del navegador abiertas sin problema, un monitor con panel IPS, para tener la mejor reproducción de colores posibles a la hora de ver una peli o un vídeo, también bastante barato para lo que se quiere hacer y una fuente de alimentación 80 plus bronze.

Comprobemos ahora algunas builds para gaming.

Como, obviamente, no puedo mostrar los resultados de las 300 combinaciones posibles, mostraré algunos interesantes.

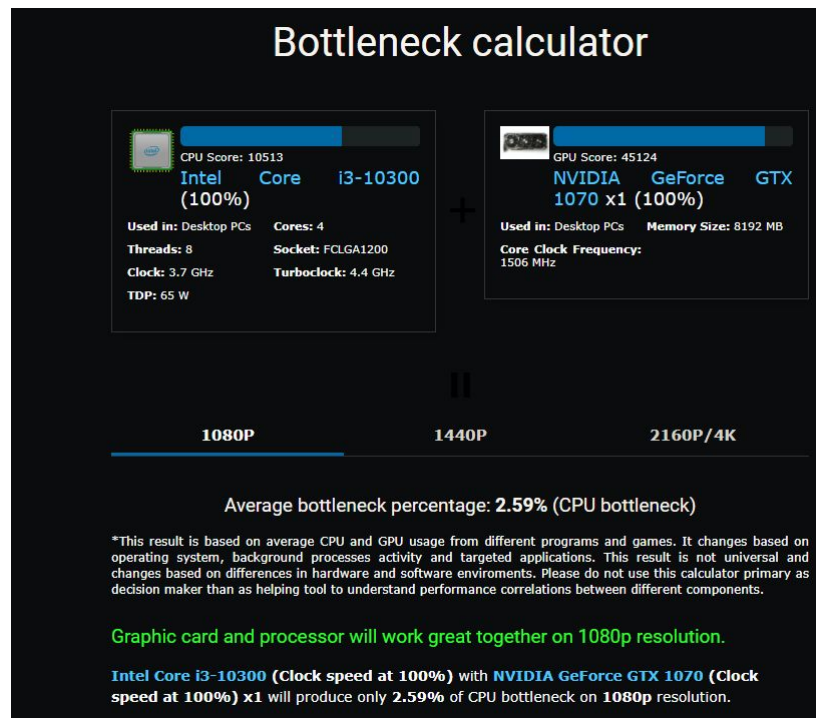
Para el tipo de pc más común, para jugar a juegos exigentes, con todo al máximo a 60 fps estables se nos recomienda lo siguiente:

RECOMENDACIÓN: 1113€

|  |  |   |  |  |  |
|--|--|---|--|--|--|
| <br>GTX 1070<br>297€                | <br>Intel Core i3-10300<br>149€ | <br>MSI MPG Z490 GAMING PLUS<br>169€ | <br>1&1&Kingston HyperX Fury Black<br>61€ |  |  |
| <br>Asus TUF Gaming VG259qM<br>379€ |  |   |  | <br>Aerocool Lux<br>58€ |  |

Como podemos ver en los puntos inferiores, hay bastantes recomendaciones.

En esta concretamente podemos ver que se nos recomienda un i3 de décima generación, concretamente el 10300 junto con una tarjeta gráfica de hace 2 generaciones, la 1070, una gráfica muy potente. Si bien puede parecer que el procesador se queda muy corto, podemos comprobar rápidamente cómo de válida es esta build buscando en una página llamada the bottleneck. Esta página calcula cómo de bien funcionan gráfica y procesador juntos y el cuello de botella que tiene una build.



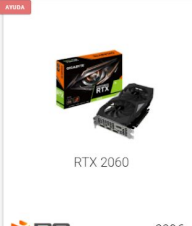


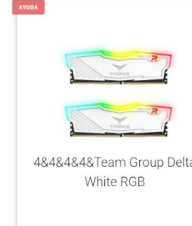
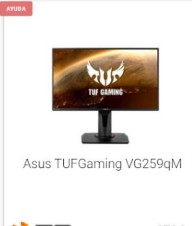
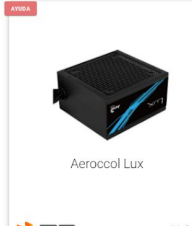
Como podemos ver, el cuello de botella es 2.59%, prácticamente insignificante, lo que nos indica que es una build muy buena para lo que queremos. Ninguna de las recomendaciones de gaming supera el 5% de cuello de botella.

En esta build contamos también con un monitor IPS, 1080p compatible con G-sync, ya que nuestra gráfica es de NVidia, 16 GB de RAM, una fuente de alimentación 80 plus bronze.

Veamos otra de las builds recomendadas para esta configuración

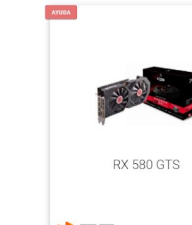


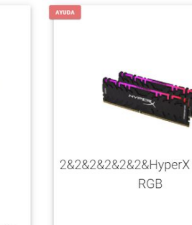
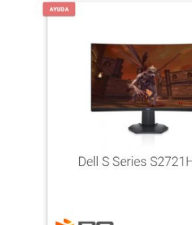
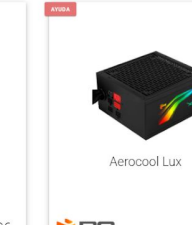
Esta recomendación es realmente buena, con una tarjeta gráfica de la generación anterior de una potencia parecida a la 1070, un ryzen 5 de generación 3000 muy eficiente para el precio que cuesta.

RECOMENDACIÓN: 967€

|  |  |  |   |
|--|--|--|---|
|  <p>RTX 2060</p> <p>329€</p>                |  <p>AMD Ryzen 5 3400G</p> <p>139€</p> |  <p>ASUS PRIME A320M-K</p> <p>54€</p> |  <p>4x4x4 Team Group Delta White RGB</p> <p>25€</p> |
|  <p>Asus TUF Gaming VG259qM</p> <p>379€</p> |  |  <p>Aerocool Lux</p> <p>41€</p>       |   |

La siguiente, una configuración más económica, muy válida, con una tarjeta gráfica algo menos potente que las anteriores 2, pero perfecta para alcanzar los fps y calidad que queremos en todos los juegos y con un monitor con Freesync, ya que nuestra gráfica es de AMD.

RECOMENDACIÓN: 795€

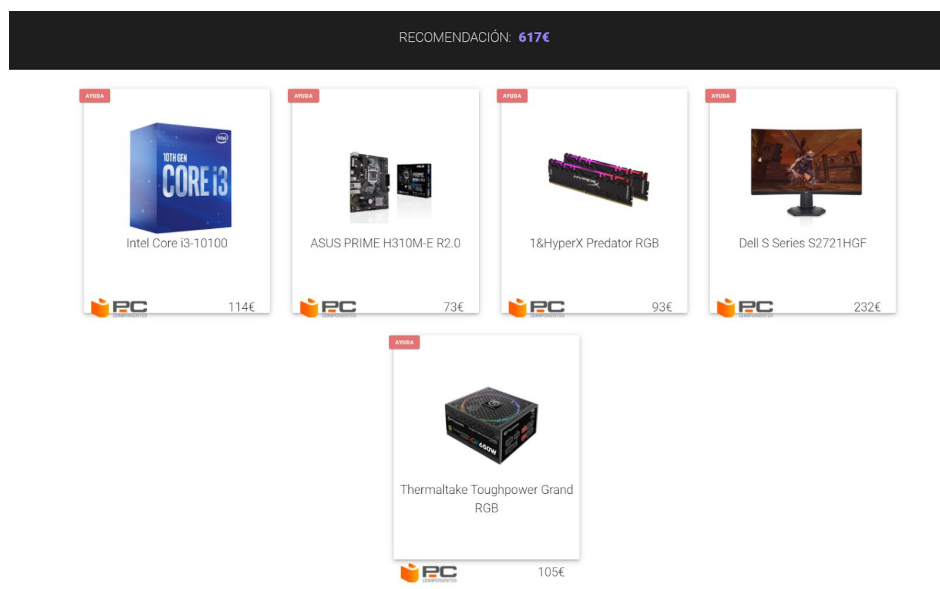
|   |  |  |  |
|---|--|--|--|
|  <p>RX 580 GTS</p> <p>219€</p>             |  <p>AMD Ryzen 5 3400G</p> <p>139€</p> |  <p>ASUS PRIME A320M-K</p> <p>54€</p> |  <p>2x8x2 HyperX Predator RGB</p> <p>93€</p> |
|  <p>Dell S Series S2721HGF</p> <p>232€</p> |  |  <p>Aerocool Lux</p> <p>59€</p>       |  |



Veamos ahora algún caso poco favorable. Empezaremos con un ordenador gaming con un nivel de esfuerzo demasiado bajo, y veamos qué se nos recomienda.



Contamos con un procesador potente, con gráficos integrados, ya que para lo que necesitamos no hace falta una gráfica dedicada, 8GB de RAM.



Como vemos, se ajusta a lo que necesitamos, aunque se parece a un ordenador de multimedia, ya que el esfuerzo de uno y otro son muy similares.

Vamos ahora al caso contrario, una build demasiado exigente, de forma que no haya hardware que pueda cumplir con los requisitos en la base de datos. El red dead redemption 2 es un juego muy exigente y muy mal optimizado, por lo que jugar a 60 fps a resolución 4K es inviable con el hardware con el que contamos.



Como vemos, justo debajo de “RECOMENDACIÓN”, se nos muestra un mensaje en rojo que nos indica que lo que queremos hacer es demasiado exigente para el hardware de nuestra base de datos, pero que las siguientes builds se le puden acercar. Como podemos ver, tenemos una RTX 2080, que es una de las tarjetas gráficas más potentes del mercado, junto con un i5-10600K, que va a 4,1GHz y es una bestia para gaming. Como queremos jugar a 4K, el monitor tiene un panel VA y una resolución 4K, con sincronización g-sync, ya que la tarjeta es de NVidia. 16GB de RAM y una fuente de alimentación 80 plus gold.

RECOMENDACIÓN: **2209€**

No hemos encontrado nada lo suficientemente potente en nuestro sistema, pero puede que esto se acerque

| Componente                       | Precio |
|----------------------------------|--------|
| RTX 2080 Super                   | 829€   |
| Intel Core i5-10600K             | 254€   |
| MSI MPG Z490 GAMING PLUS         | 169€   |
| Kingston HyperX Fury Black       | 61€    |
| Acer Predator XB273GPbmiiiprx    | 791€   |
| Thermaltake Toughpower Grand RGB | 105€   |

Como hemos podido ver, a pesar del input, el programa siempre recomendará algo. Esto está motivado por la sensación que tengo como usuario de Netflix en

contraposición de la sensación al ser usuario de HBO. Cuando en HBO buscas una película que no tienen, la pantalla se queda en negro y te pone un mensaje indicando que dicha película no están en su base de datos. Por otra parte, Netflix, si no tiene la película que quieres, te muestra otras que pueden interesarte, por ser parecidas a la que buscabas. En ninguno de los dos casos has obtenido lo que querías, pero en uno de ellos te has sentido escuchado. De alguna forma notas que la página se preocupa por ti y por satisfacer tus necesidades. Casi parece una persona, en vez de un programa, a diferencia de HBO.

Inspirado por esto, he programado la aplicación para que también escuche al usuario y siempre le de una alternativa, avisando de que lo que él quiere no puede darse.

## 4.5 Análisis DAFO

Voy a realizar ahora un análisis crítico del proyecto después de todo este tiempo de desarrollo, viendo lo que tenía en mente en un principio y lo que ha acabado siendo, desarrollando un poco el punto en el que estoy. Esto lo voy a hacer mediante un análisis DAFO (Debilidades, Amenazas, Fortalezas, Oportunidades).

**Debilidades:** Mi aplicación no aprovecha al 100% las ventajas que ofrecen las bases de datos de grafos de conocimiento como Neo4j. Si bien es cierto que a una mayor escala, la aplicación funcionaría mucho más rápido en este tipo de bases que en una normal, a pequeña escala, como es el caso, la diferencia es imperceptible. Además, al ser una aplicación sin usuarios ni cookies ni nada por el estilo, las recomendaciones son objetivas, por lo que tampoco aprovecha esa búsqueda de patrones de la que tanto se habla cuando se trata el tema de este tipo de bases de datos. Estoy contento con el algoritmo desarrollado para gaming, me parece que funciona y devuelve unos resultados muy buenos y aunque los resultados de edición y las demás, también son acertados, el algoritmo de estas es más simple y se podría perfeccionar mucho. Por último, la base de datos tiene pocas relaciones, por lo que, de nuevo, no se explota al máximo sus capacidades.

**Amenazas:** La principal amenaza fue conseguir una puntuación para procesador y gráfica que se ajustase a la realidad, ya que al final unos serán mejores que otros en ciertas tareas y peores en otras, por lo que resulta complicado decir de forma objetiva cuál es mejor, habiendo poca concordancia en general en la red. Finalmente encontré una página que más o menos parecía representar bien la realidad y que menciono en la bibliografía. Encontrar un algoritmo fiel para

recomendar ordenadores gaming también fue una tarea complicada, sin poder ayudarme de ninguna búsqueda por google, teniendo solo mi lógica.

**Fortalezas:** Los algoritmos utilizados son bastante precisos, dando (en gaming) resultados de builds con un máximo de un 5% de cuello de botella, lo que está muy por encima de lo que pensé que conseguiría. Aclaro la actividad gaming, debido a que en actividades como edición, un cuello de botella por parte de la gráfica no es problemático, ya que esta tendrá poca carga computacional, pues la mayor parte depende del procesador.

Por otra parte creo que la interfaz que he hecho, tanto input como output, aunque especialmente input es agradable a la vista, intuitiva y proporciona un buen feedback al usuario.

**Oportunidades:** Si continuase perfeccionando mi trabajo, en poco tiempo podría utilizarse para tener una idea clara y precisa de qué ordenador comprarse y con el tiempo, llegar a ser un buen complemento para empresas de venta de hardware, como la ya mencionada PcComponentes y cualquier otra que esté interesada.

## 4.6 Líneas de futuro

## 4.6 Conclusiones

No es lo mismo trabajar con códecs muy comprimidos

Si estás empezando, lo más seguro es que grabes con codecs h.264 y h.265, por lo que el proceso de decodificación, lo va a llevar en su mayor parte, la CPU.

Mínimo i7 8 núcleos.

Aceleración por GPU, o sea, ayuda.

8GB de VRAM.

## Bibliografía o referencias:

<https://www.nategentile.com/>

<https://www.youtube.com/channel/UC36xmz34q02JYaZYKrMwXng>

<https://www.youtube.com/channel/UCJKFZ26gRK7AJXfmuxWR9Cg>

[https://www.youtube.com/channel/UC40Ztmc\\_1l1euUR-tHh\\_irQ](https://www.youtube.com/channel/UC40Ztmc_1l1euUR-tHh_irQ)

<https://www.pccomponentes.com/>

[https://www.youtube.com/watch?v=jaDfWL7bPbE&t=188s&ab\\_channel=RubenGuo](https://www.youtube.com/watch?v=jaDfWL7bPbE&t=188s&ab_channel=RubenGuo)

<https://www.chamanexperience.com/video/tipos-de-codecs-y-formatos-de-video/>

<https://gamerpc.es/configuraciones/pc-para-edicion-de-video/>

