

Sistemas de Información de Gestión y Business Intelligence

Universidad de León

Memoria de la aplicación

Índice

1.Descripción del problema	3
2.Herramientas	4
3.Aplicación	7
3.1.Base de datos	7
3.2.Backend y Frontend	9
4.Algoritmo de recomendación	21
4.1.Explicación	21
5.Observación de resultados	28
6.Análisis	31
6.1.DAFO	32
7.Líneas de futuro	33
8.Lecciones aprendidas	34
9.Referencias y bibliografía	34
9.1.Referencias	34
9.2.Bibliografía	35

1.Descripción del problema

El tema sobre el que he desarrollado el proyecto es sobre rutas de senderismo, y es que a mi me encanta hacer una escapada de vez en cuando para recorrer alguno de los cientos de caminos que hay en nuestra comunidad, descubriendo lugares que merece la pena ver y disfrutar.

Pero siempre que he ido a buscar un nuevo sitio al que ir, a pesar de haber muchas web en las que nos informan de las distintas rutas existentes, no hay ningún sitio en el que se haga una recomendación más personal, teniendo en cuenta los gustos de cada persona. Es por eso que me llamó la atención este tema, con el fin de crear un sistema en el que poder registrarnos, guardar nuestras rutas favoritas y poder escoger, en función de nuestros gustos, entre todas las rutas registradas en la base de datos. En la imagen siguiente vemos cómo al buscar alguna ruta en google, lo que nos muestra es un top de rutas escogidas por la opinión pública, que no tiene por que ser lo que le guste a todo el mundo.



Ejemplo de búsqueda de recomendación de rutas

El principal problema con el que me topé es que no existe ninguna base de datos sobre esta temática. Por lo que el primer paso fue buscar por muchos lugares toda la información de cada ruta y juntarla toda en un solo lugar. Después pensé en crear usuarios para que así cada uno, como dije anteriormente, pueda guardar las rutas que más le hayan gustado. Y una vez hecho todo esto, me puse manos a la obra con el algoritmo de recomendación, pero esto lo comentaremos más adelante en las sucesivas secciones.

2.Herramientas

Antes de comenzar a hablar sobre las distintas herramientas que he usado para desarrollar el proyecto, me gustaría recordar que para poder realizar dicho trabajo ha sido necesario aprender a utilizar la nueva tecnología destinada a crear bases de datos orientadas a grafos, Neo4j. En [mi cuaderno OSF](#) se encuentra una explicación de cómo funciona y cómo ha sido mi relación con este entorno en un primer momento.

Además, he echado un vistazo al libro Graph Algorithms ¹ , el cual nos introduce en el mundo de los algoritmos que podemos crear gracias a Neo4j. En los apartados posteriores de este documento explicaré un algoritmo que yo he “inventado” y que nos servirá para que las recomendaciones sean aún más personales y acertadas.

Pero claro, un sistema que ofrece recomendaciones es necesario que disponga de una base de datos, como el lógico. En mi caso, debido a la falta de información sobre el tema que escogí, me encontré con que no había por ningún lado ninguna base de datos sobre las rutas que se pueden hacer en la comunidad de Castilla y León. Es por esto que la base de datos la he realizado yo desde cero. Para esto, tuve que aprender el lenguaje de estas bases de datos, Cypher. Cuando estaba realizando la base de datos, casi siempre utilizaba las mismas sentencias, es por eso que aprovecho para destacar que en el apartado de Cypher de mi cuaderno de OSF, he realizado una pequeña guía de este lenguaje con las operaciones fundamentales y más importantes que he tenido que aprender y utilizar para poder desarrollar correctamente la base de datos. Dichas operaciones eran monótonas y se repetían a lo largo de la creación de la base de datos. Más adelante pondré un ejemplo de algunas líneas de código.



Graph Database

&
Cypher

Una vez tenía la temática clara y ya había conseguido tener la base de datos creada, el siguiente paso era decidir dónde íbamos a crear el sistema de recomendación. La primera idea fue realizar una aplicación móvil, pero la documentación para poder conectar la app con la base de datos de Neo4j era muy escasa. Por lo que finalmente me decidí por hacer una aplicación web.

Escogí esta opción sobretodo por los conocimientos que tenía para desarrollarla, ya que el año pasado tuve que hacer un trabajo en el cual programamos, mi equipo y yo, una aplicación web con estas tecnologías.

En este punto ya tenía claro cómo empezar, pero me faltaba algo muy importante, que era conectar dicha aplicación con Neo4j. Para ello necesitaba un driver, que no es más que un conjunto de librerías destinadas a facilitarnos el trabajo permitiéndonos mandar y recibir información de la base de datos. Investigué un poco y encontré el driver de Neo4j ², que fue fundamental para el correcto desarrollo de la aplicación.

Pero con está documentación no fui capaz de comprender al completo como podría crear un backend funcional capaz de conectarse a Neo4j. Es por eso que investigué en github sobre temas relacionados con está tecnología. Rápidamente encontré una persona que había subido una documentación muy útil sobre esto. Aquí dejo su github ³ para que se pueda visitar. En este github podemos encontrar muchas cosas, entre otras un ejemplo de código ⁴ en javascript de como conectarnos a Neo4j.

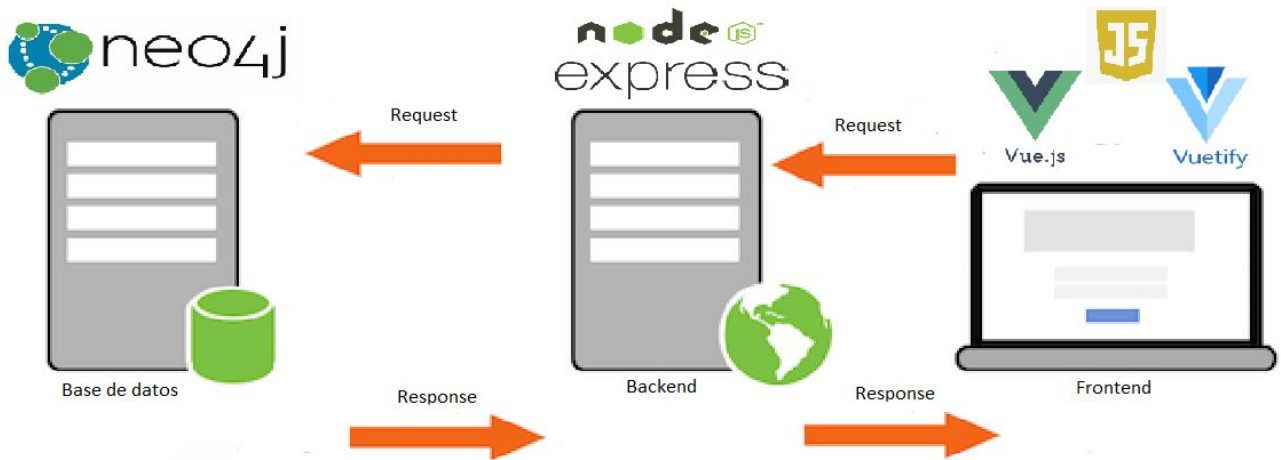
La aplicación web se compone principalmente de dos partes, frontend y backend. A continuación vemos una breve definición de qué es esto junto con las tecnologías que he necesitado para poder desarrollar cada una de ellas:

- Frontend. Es la parte visual de la aplicación, la interfaz con la que el usuario interactúa.
 - Vue.js ⁵ : Nos permite crear la aplicación, es la estructura de la misma.
 - Vuetify ⁶ : Con esto realizamos la interfaz de la app, ya que aporta un montón de complementos.
 - Javascript ⁷ : Lenguaje necesario para poder programar el frontend.
 - Axios ⁸ : Es un cliente HTTP para javascript, me permitió realizar las consultas al backend.
 - Electron ⁹ : Nos permite transformar la app web en una de escritorio, evitando así tener que abrir el navegador.
- Backend. Es la parte que no vemos de todo el funcionamiento, encargada de comunicarse con la base de datos y así poder traernos de vuelta toda la información.
 - Node.js ¹⁰ : Nos da el esqueleto del backend.
 - Express ¹¹ : Con esto seremos capaces de recibir consultas del frontend.
 - Javascript ¹² : Lenguaje necesario para poder programar el frontend.
 - Driver Neo4j ¹³ : Nos permite realizar consultas de Cypher a la base de datos para extraer información, así como guardar nuevos datos.
- Base de datos. El lugar donde se almacenan todos los datos, donde están guardadas todas las rutas y sus características, los usuarios y sus relaciones con sus favoritas, etc.
 - Neo4j ¹⁴ : Es un software libre de base de datos orientada a grafos, implementado en Java.
 - Cypher ¹⁵ : Lenguaje usado por Neo4j para la consulta a su base de datos o para la creación de la misma.

Por último, comentar que la programación de la aplicación y todo lo que conlleva la he realizado en Visual Studio Code ¹⁶ , un editor de código muy completo y fácil de usar.

3.Aplicación

La imagen que vemos a continuación representa la estructura y funcionamiento de mi aplicación:



El funcionamiento es muy sencillo. Cuando un usuario quiere conocer las rutas más espectaculares de Castilla y León lo que hará será registrarse en la aplicación. El frontend, cuando el usuario clicca en el botón registrar, manda una petición al backend, y este realiza una consulta a la base de datos para crear un nuevo nodo que represente un nuevo usuario. Una vez registrado y dentro de la aplicación pondrá los parámetros necesarios para obtener una recomendación, entonces, el frontend vuelve a mandar una petición al backend para solicitar una ruta que cumpla con las características especificadas. El backend realiza la consulta a la base de datos de Neo4j y devuelve los datos oportunos al frontend, donde se muestran al usuario por medio de la interfaz.

Ahora profundizaremos más en los componentes de la aplicación. Pasamos a hablar de la base de datos, donde explicare que guardo en cada nodo, que nodos tengo, como cree la bbdd de cero, mostrando alguna parte de las sentencias que usé.

3.1.Base de datos

Como ya explique anteriormente, yo creé mi propia base de datos. Esto se debe a que a pesar de existir una gran cantidad de aportaciones en Kaggle ¹⁷, ninguna se adapta a lo que yo buscaba. Todas son muy genéricas y no se especializan en muchas cosas, sobretodo se tratan temas globales, videojuegos, películas, etc. Tratan de temas globales, y lo que yo pretendía era hacer algo más personal, más relacionado con mi tierra y con mi gustos. Es por eso que decidí crear la base de datos por mi cuenta y desde cero.

Lo que hice fue crear un documento (el cual se encuentra en github) en el que escribí los comandos y toda la información necesaria para que, una vez acabado, lo pegara en el Neo4j browser y se creara correctamente la base de datos.

Los nodos principales que componen la bbdd son las rutas, de las cuales guardo la siguiente información:

- **Nombre:** Identificador de la ruta, nos permite saber como se llama.
- **Distancia:** Para saber de antemano cuantos kilómetros la componen.
- **Tiempo:** Número de horas que nos llevará realizar el recorrido.
- **Punto de salida:** Lugar desde donde se empieza la ruta, para poder localizarla en un mapa.
- **Desnivel:** Metros que subimos a lo largo de la ruta.
- **Circular:** Nos dice si el recorrido empieza y acaba en el mismo lugar, o si por el contrario empieza en el lugar contrario al que termina.
- **Imagen:** Una dirección web a la imagen del recorrido plasmado en un mapa. Hay rutas de las que no encontré su mapa, por lo que se mostrará otra imagen relacionada con el paseo.

Además, los nodos de las rutas que acabamos de comentar están unidos a otros nodos para aportar aún más información. Diferentes nodos:

- **Época:** Nos indica en qué momento del año será más apropiado realizar el camino. Se divide en:
 - Verano
 - Invierno
 - Otoño
 - Todas

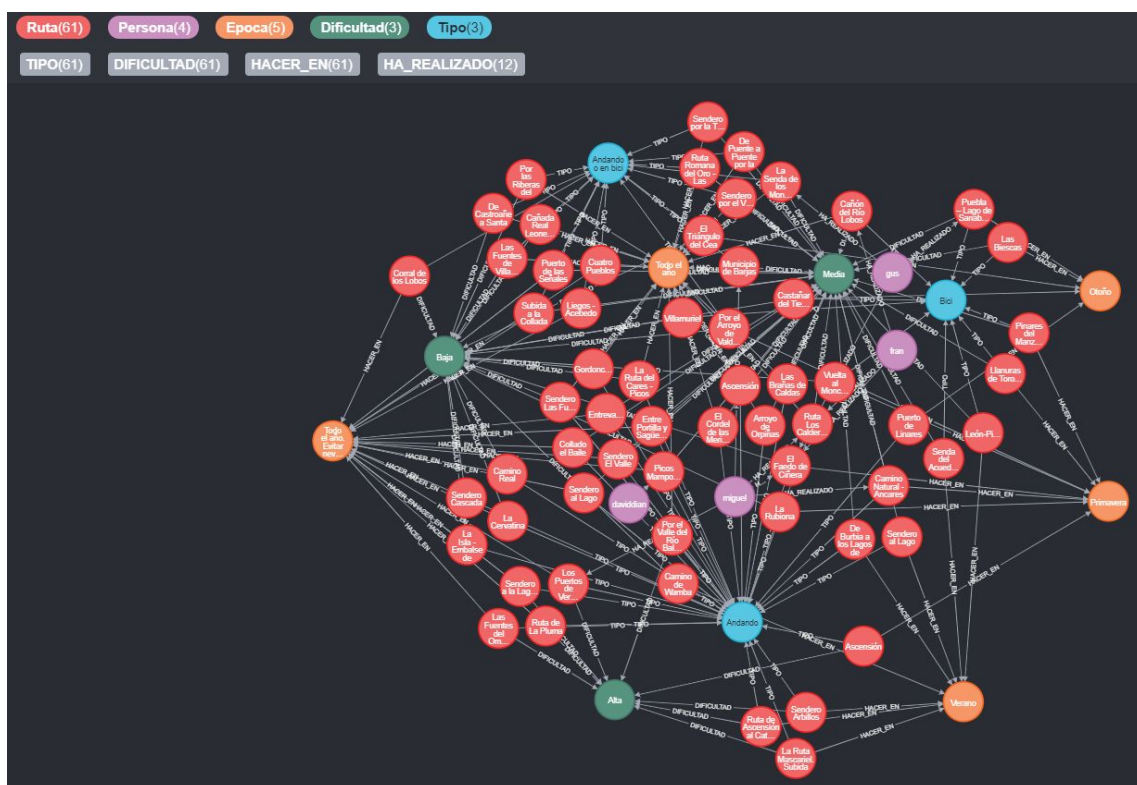
(No hay invierno ya que ninguna ruta se recomendaba hacer exclusivamente en ésta época)

- **Dificultad:** Informa sobre el nivel de la ruta, con el fin de que una persona novata en realizar rutas, no realice una de una dificultad demasiado alta para su experiencia. Hay tres tipos:
 - Alta
 - Media
 - Baja
- **Tipo:** Esto nos dice si la ruta está pensada para realizarse exclusivamente andando, en bici o se puede realizar de cualquiera de las dos formas. Por tanto, los nodos serán de tres tipos:
 - Andando
 - En bici
 - Andando o en bici

Por otro lado, ya que en mi aplicación será necesario disponer de un usuario y contraseña para poder acceder al sistema, habrá otro tipo de nodo, el de las personas. Dentro de estos se almacena la siguiente información:

- **Nombre**
- **Apellidos**
- **Nombre de usuario**
- **Año de nacimiento**
- **Contraseña.**

La imagen siguiente muestra cómo queda la base de datos una vez se han creado todos los nodos y todas las relaciones:



3.2.Backend y Frontend

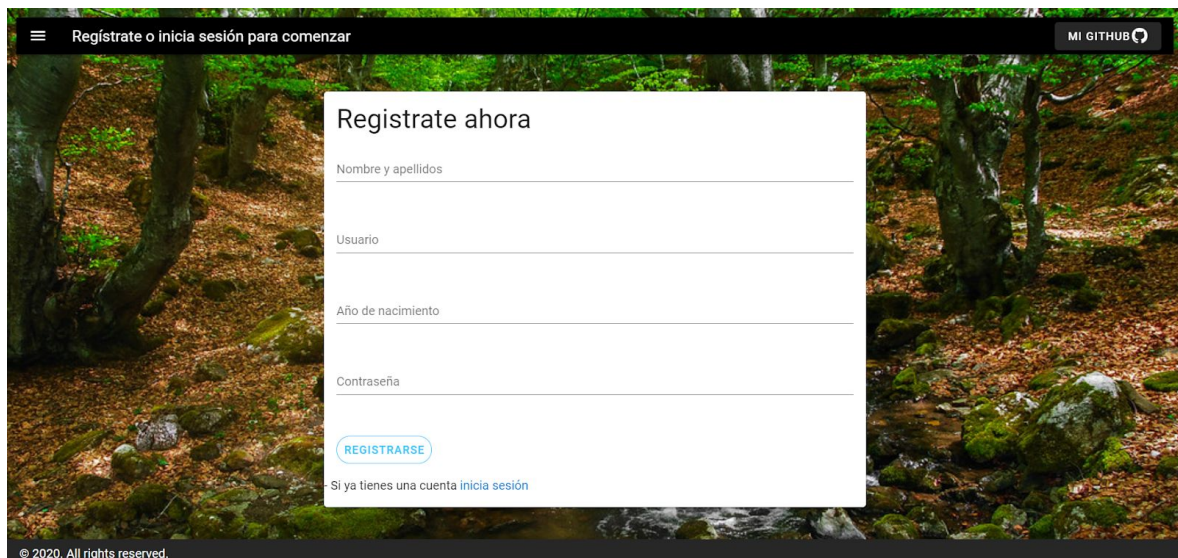
Como expliqué antes, el funcionamiento de la aplicación consiste básicamente en peticiones realizadas desde el frontend al backend solicitando la información necesaria de cada momento, y después es el backend el que realiza la conexión con la base de datos Neo4j para pedirle todos los datos.

Para crear el backend lo primero que hice fue crear una carpeta, llamada “appbackend” donde se aloja el archivo javascript principal (y único), “app.js”. Después, dentro de la carpeta ejecuté el comando “npm init” con el que se crean los ficheros “package.json”. Gracias a estos se llevará a cabo un registro de todas las

dependencias que vayamos instalando en nuestro backend, para que así si otra persona lo quiere ejecutar en su ordenador pueda descargar lo necesario para que funcione correctamente.

Para poder realizar el algoritmo que tenía en mente y que la experiencia con la aplicación de cada persona fuera personal y adaptada a cada uno, decidí crear usuarios. Es decir, para poder entrar a la aplicación es necesario disponer de una cuenta.

En la siguiente imagen vemos como cuando se inicia la aplicación se nos pide que nos registremos, o bien, si ya tenemos una cuenta que iniciemos sesión. Además, arriba a la derecha podemos ver un enlace a [mi Github](#) para visualizar el código fuente de la aplicación:

The image shows a web application interface for user registration. At the top, there is a dark header bar with a hamburger menu icon on the left, the text "Regístrate o inicia sesión para comenzar" in the center, and a "MI GITHUB" link with a GitHub logo on the right. The main content area features a large, semi-transparent white registration form centered over a background image of a forest floor with trees and fallen leaves. The form is titled "Regístrate ahora" and contains four input fields: "Nombre y apellidos", "Usuario", "Año de nacimiento", and "Contraseña". Below these fields is a blue button labeled "REGISTRARSE". At the bottom of the form, there is a link that says "Si ya tienes una cuenta inicia sesión". A small copyright notice "© 2020. All rights reserved." is visible in the bottom left corner of the page.

Frontend: Registro

Cuando el usuario rellena los campos que se le piden en la interfaz para poder registrarse, son enviados al backend. Lo primero que hacemos en el frontend es crear la petición pasándole los datos:

```
// Make a request for a user with a given ID
axios
.post('http://localhost:3000/registro', {
  usuario: this.usuario,
  nombre: this.nombreapellidos,
  contraseña: this.contrasena,
  anonacimiento: this.anonacimiento
})
```

Frontend: Envío de datos

Una vez enviados los datos al backend serán guardados en unas variables, las cuales usaremos después para completar la consulta de Cypher. De esta forma, en la base de datos habremos registrado un nuevo usuario.

En la imagen siguiente podemos ver el código que realiza lo que acabamos de explicar:

```
app.post("/registro", function (req, res) {
  var usuario = req.body.usuario;
  var nombre = req.body.nombre;
  var contrasena = req.body.contrasena;
  var anonacimiento = req.body.anonacimiento;
  var query = "CREATE (n:Persona{nombre:'" + nombre + "', nacimiento:" + anonacimiento + ", "
  +"usuario:'" + usuario + "', password:'" + contrasena + "'})";

  const session = driver.session();

  const resultPromise = session.run(query);
  resultPromise.then(result => {

    res.json({
      msg: 'Bien'
    });
    session.close();
  })
  .catch((error) => {
    // handle error
    res.json({
      msg: 'Error'
    });
    console.log(error)
    session.close();
  })
});
```

Backend: Registro de un usuario

Como vemos en la imagen anterior, abrimos una sesión de NEO4J y le enviamos la query, con la sentencia 'session.run(query)'. A partir de aquí se crea una promesa, donde guardamos el resultado de la ejecución de la query. Tenemos dos opciones, que se ejecute sin problemas y estaremos en el caso ".then" donde devolveremos un mensaje de que ha ido bien, o por el contrario que por alguna razón se produzca un error, y entraremos en el caso de ".catch", mandando así un mensaje de error al frontend.

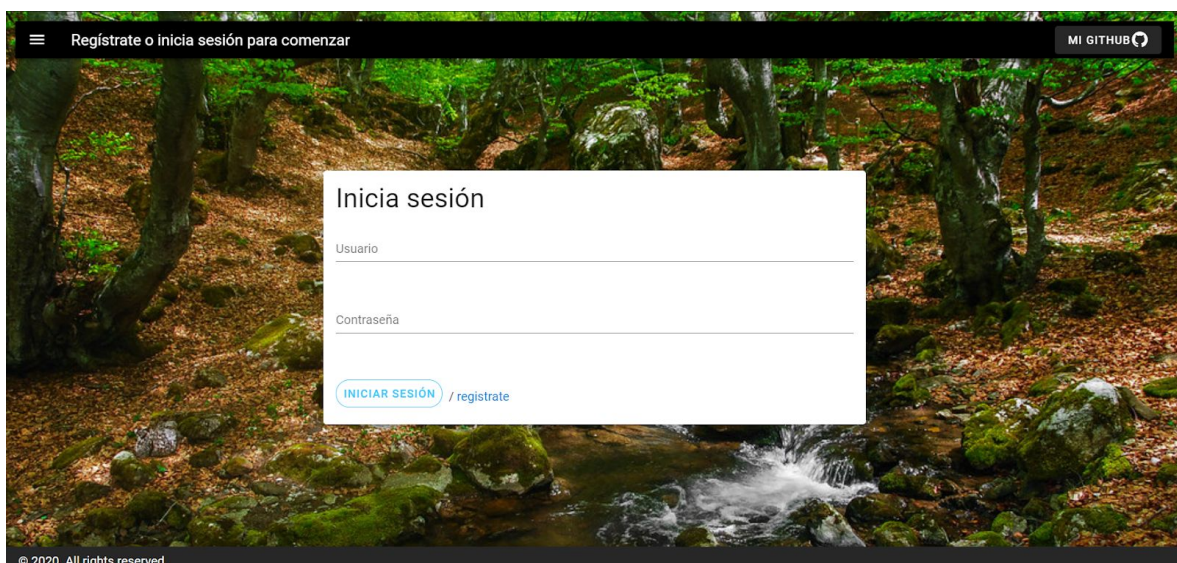
Una vez se ha ejecutado la consulta anterior y todo ha ido bien (o mal) en el frontend cambiamos de vista y nos dirigimos a la página principal de la aplicación, y si todo ha ido mal se mostrará un mensaje de error por pantalla.

El código que se encarga de ver como ha ido la petición es el siguiente:

```
.then((response) => {  
  // handle success  
  var json = {msg: 'Error'};  
  
  if(JSON.stringify(response.data)==JSON.stringify(json)){  
    alert('El nombre de usuario ya está uso')  
    this.limpiarFormulario()  
  }else{  
    this.$router.push({ name: 'Home', params: { usuario: this.usuario } })  
    this.limpiarFormulario()  
  }  
})
```

Frontend: Resultados de registro

Para iniciar sesión si ya tenemos una cuenta creada en la aplicación, enviaremos nuestro nombre de usuario y contraseña al backend. Una vez allí lo que haremos será preguntar a la base de datos si hay algún usuario con ese nombre, y si es así devolveremos el nombre de usuario, y si no se encuentra o la contraseña es incorrecta, enviaremos un mensaje de error. Lo primero, el usuario deberá rellenar el siguiente campo en la interfaz para poder enviar los datos al backend:

The image shows a web application interface for logging in. At the top, there is a navigation bar with a hamburger menu icon, the text "Regístrate o inicia sesión para comenzar", and a "MI GITHUB" link. The main content area features a large background image of a forest stream. Overlaid on this is a white login form titled "Inicia sesión". The form contains two input fields: "Usuario" and "Contraseña". Below these fields are two buttons: a blue button labeled "INICIAR SESIÓN" and a link labeled "/ regístrate". At the bottom left of the image, there is a small copyright notice: "© 2020. All rights reserved."

Frontend: Inicio de sesión

Una vez introducidos los datos, estos serán procesados como he explicado anteriormente, y el código encargado de ello es el siguiente:


```

app.post("/login", function (req, res) {
  var usuario = req.body.usuario;
  var contrasena = req.body.password;
  var query = "MATCH (n:Persona) WHERE n.usuario='" + usuario + "' AND n.password='" + contrasena + "' return n";

  const session = driver.session();

  const resultPromise = session.run(query);
  resultPromise.then(result => {

    if (result.records.length == 0) {
      res.json({
        msg: 'Error'
      })
    }
    else {
      var record = result.records[0].get(0).properties.usuario;
      res.send(record);
    }

    session.close();
  })
  .catch((error) => {
    // handle error
    res.json({
      msg: 'Error'
    });
    console.log(error)
    session.close();
  })
});

```

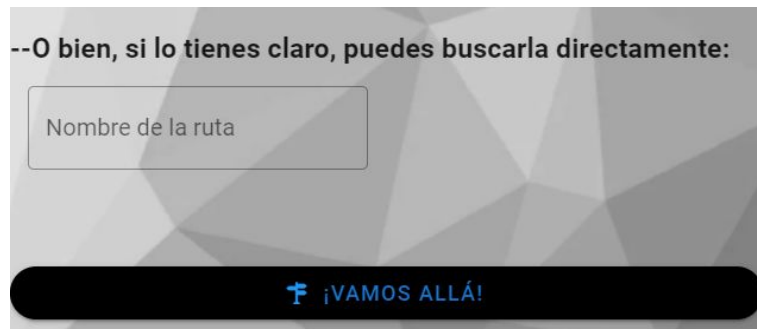
Backend: Inicio de sesión

Como vemos, utilizamos el usuario y la contraseña que nos llegan para completar la query. Una vez ejecutada se comprueba si obtenemos cero resultados devolviendo error en este caso, mientras que si el resultado tiene datos, quiere decir que ha encontrado al usuario.

Una vez se ha iniciado sesión correctamente, el usuario se encontrará en la página principal de la aplicación:

Frontend: Página principal

Una vez en la página principal, la persona podrá buscar una ruta directamente por su nombre si lo tiene muy claro o si quiere comprobar qué rutas hay registradas en la base de datos (a pesar de no haber muchas, la idea es ir añadiendo cada vez más y más rutas de todas las variedades y tipos). El “centro de mando” de la aplicación es donde encontramos la opción de introducir un nombre:

A screenshot of a web application's search interface. At the top, there is a text prompt: "--O bien, si lo tienes claro, puedes buscarla directamente:". Below this is a text input field with the placeholder text "Nombre de la ruta". At the bottom of the interface is a dark blue button with a white magnifying glass icon and the text "¡VAMOS ALLÁ!".

--O bien, si lo tienes claro, puedes buscarla directamente:

Nombre de la ruta

🔍 ¡VAMOS ALLÁ!

Frontend: Búsqueda por nombre

Una vez introducido el nombre o la palabra clave, se crea una consulta de nuevo a la base de datos. Dicho texto se envía al backend, donde se creará una consulta para comprobar si hay alguna ruta en cuyo nombre se encuentre la palabra escrita por el usuario.

Código del frontend donde enviamos la palabra clave y cuando recibimos el resultado mostramos un mensaje de error al no encontrarse ninguna ruta, o bien, guardamos en un array las distintas rutas que han coincidido:

```
axios
.post('http://localhost:3000/buscarnombre', {
  nombre: this.rutaBuscarNombre
})
.then((response) => {
  // handle success
  var json = {msg: 'Error'};

  if(JSON.stringify(response.data)==JSON.stringify(json)){
    alert('No se ha encontrado ninguna ruta con ese nombre')
  }else{
    for(var i=0;i<response.data.length;i++){
      this.rutas = response.data
    }
    this.mostrarBotonG = true;
  }
})
```

Frontend: Búsqueda por nombre

Por otro lado, el código del backend es el siguiente. En primer lugar recogemos los datos que nos entran y creamos la query:

```
app.post("/buscarnombre", function (req, res) {  
  var nombre = req.body.nombre;  
  var rutas = [];  
  var query = "MATCH (r:Ruta)-[:HACER_EN]->(e), (r)-[:DIFICULTAD]->(d), (r)-[:TIPO]->(t) "  
    + "WHERE TOLOWER(r.nombre)=~TOLOWER('."+nombre+".*') "  
    + "RETURN r.nombre, r.distancia, r.tiempo, r.punto_salida, r.desnivel, r.circular, r.imagen, e.nombre, d.nombre, t.nombre";
```

Backend

Después, una vez ejecutada la query guardamos los distintos datos de las rutas (si los hubiere) en un array para enviárselos al frontend y que éste pueda mostrarlos:

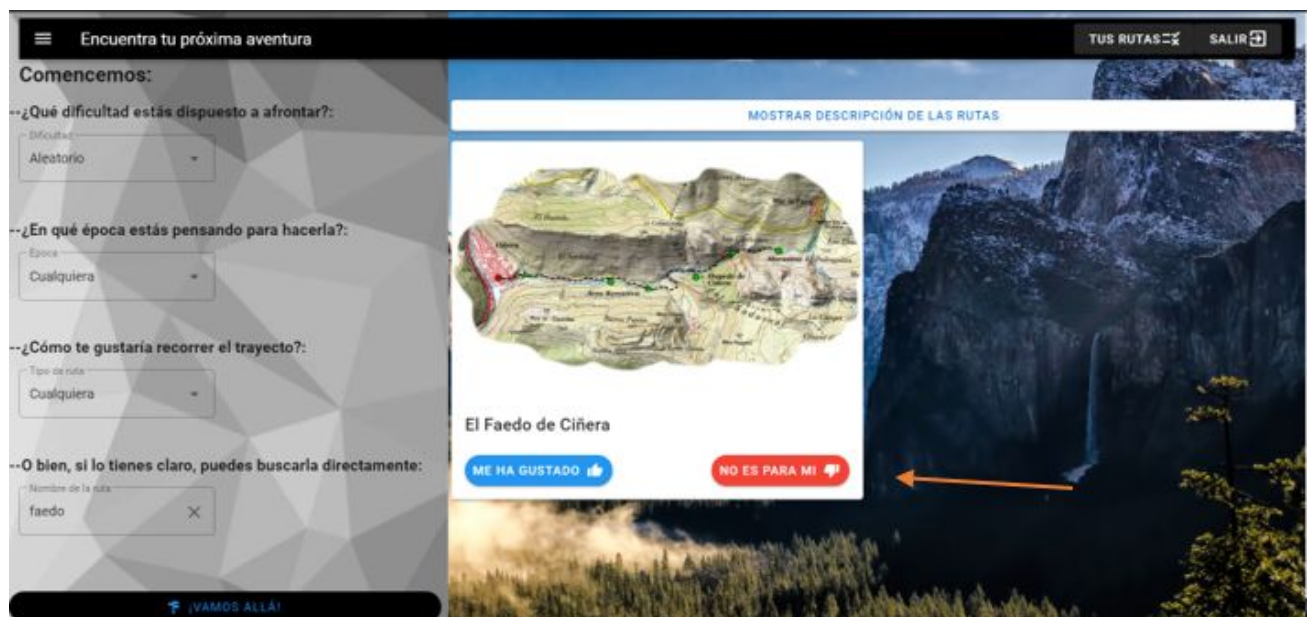
```
const resultPromise = session.run(query);  
resultPromise.then(result => {  
  
  if (result.records.length == 0) {  
    res.json({  
      msg: 'Error'  
    })  
  }  
  else {  
    for(var i=0;i<result.records.length;i++){  
  
      var ruta = {  
        nombre_ruta: result.records[i]._fields[0],  
        distancia: result.records[i]._fields[1],  
        tiempo: result.records[i]._fields[2],  
        punto_salida: result.records[i]._fields[3],  
        desnivel: result.records[i]._fields[4],  
        circular: result.records[i]._fields[5],  
        imagen: result.records[i]._fields[6],  
        epoca: result.records[i]._fields[7],  
        dificultad: result.records[i]._fields[8],  
        tipo: result.records[i]._fields[9],  
        recomendada: ''  
      }  
  
      rutas.push(ruta);  
    }  
  
    res.send(rutas)  
  }  
  
  session.close();  
})
```

Backend

En este tipo de búsqueda no se ha visto implicado ningún tipo de criterio, ya que simplemente se busca por el nombre. Pero el usuario dentro de la aplicación podrá seleccionar una serie de parámetros que han de cumplir las rutas que se le muestran. Pudiendo elegir como realizar la ruta, si en bici o andando, el nivel de dificultad de la ruta o la época más recomendada para llevarla a cabo.

Una vez introducidos los datos, se ejecuta un algoritmo de recomendación que explicaré en la siguiente sección de forma más detallada. De esta forma me aseguro que las rutas que se le brindan a cada persona son las más adecuadas a sus intereses y que además tienen una probabilidad muy alta de que les encanten.

Para poder llevar a cabo el algoritmo de una forma efectiva, es necesario que el usuario disponga de algunas rutas en su lista de favoritos. Para esto realicé la función de que cuando busca una ruta pueda marcarla como favorita o que no le ha gustado. Por ejemplo:



Frontend: Calificar una ruta

Al clicar en el botón de “Me ha gustado” se creará una petición al backend para que se guarde en la base de datos que esa ruta le ha gustado a ese usuario, y lo mismo si no le ha gustado.


```

axios
.post('http://localhost:3000/relacionbuena', {
  usuario: this.usuario,
  ruta: nombre_ruta
})
.then((response) => {
  // handle success
  var json = {msg: 'Error'};

  if(JSON.stringify(response.data)==JSON.stringify(json)){
    alert('Parece que ya has calificado esta ruta antes')
  }else{

    this.dialog = true;
  }
})
.catch((error) => {
  // handle error
  console.log(error);
})
.then(function () {
  // always executed
})

```

Frontend: Añadir ruta a favoritas

En el código anterior enviamos una petición al backend para añadir la ruta seleccionada a la lista de favoritas, y este lo que hace es crear la query correspondiente y ejecutarla en la base de datos, creándose así la relación [:HA_REALIZADO {experiencia:'Buena'}] entre usuario y ruta:

```

app.post("/relacionbuena", function (req, res) {
  var usuario = req.body.usuario;
  var ruta = req.body.ruta;
  var query = "MATCH (p:Persona), (r:Ruta) "
    + "WHERE p.usuario= '"+usuario+"' AND r.nombre='"+ruta+"' AND NOT EXISTS((p)-[:HA_REALIZADO {experiencia: 'Buena'}]->(r)) "
    + "AND NOT EXISTS((p)-[:HA_REALIZADO {experiencia: 'Mala'}]->(r)) "
    + "CREATE (p)-[:HA_REALIZADO {experiencia: 'Buena'}]->(r) "
    + "RETURN p, r";

  const session = driver.session();

  const resultPromise = session.run(query);
  resultPromise.then(result => {

    if (result.records.length == 0) {
      res.json({
        msg: 'Error'
      })
    }
    else {
      res.send("Bien");
    }

    session.close();
  })
  .catch((error) => {
    // handle error
    res.json({
      msg: 'Error'
    });
    console.log(error)
    session.close();
  })
});

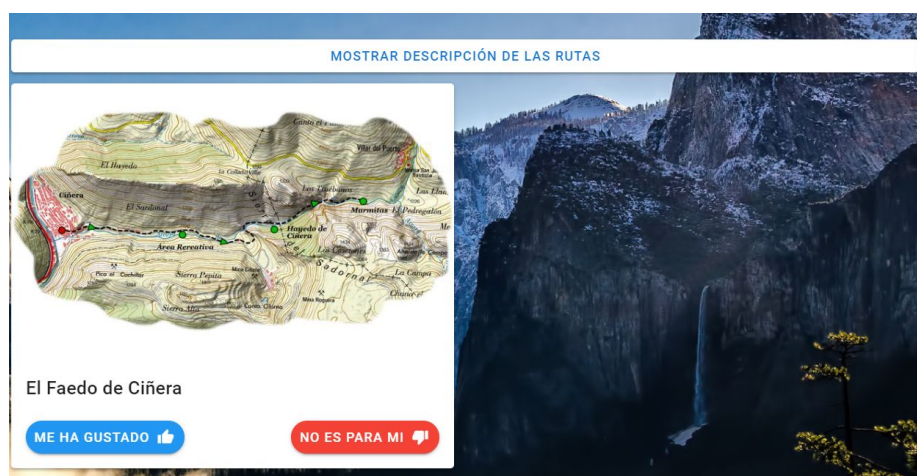
```

Backend: Añadiendo ruta a favoritas

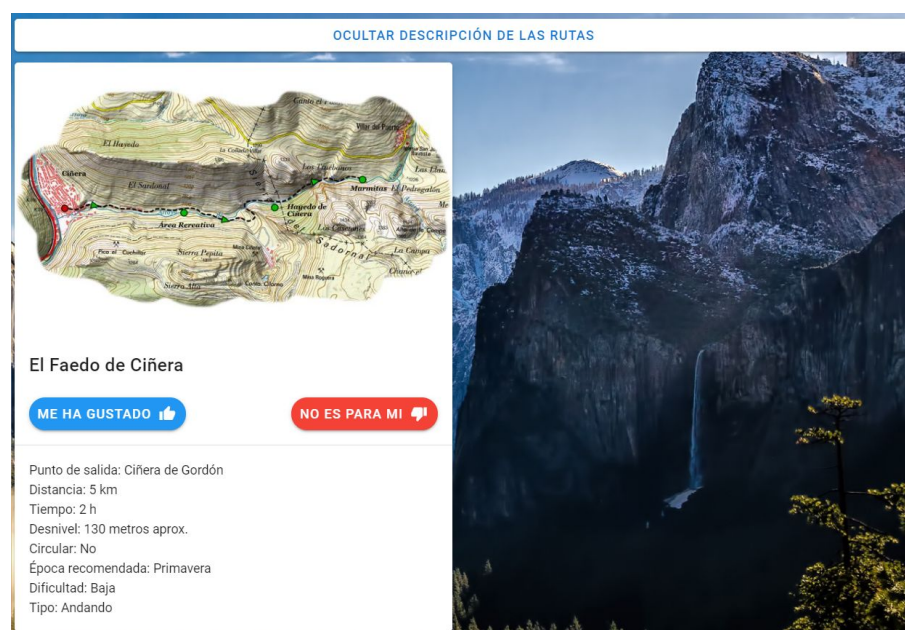
El procedimiento es similar cuando el usuario clicca en el botón “No es para mí”, solo que la experiencia de la relación pasa de “Buena” a “Mala”.

De está forma se guardará la información en la base de datos y quedará registrado que a ese usuario esa ruta no le gustó o no la disfruto lo suficiente por cualquier motivo, por lo cual no se tendrá en cuenta más adelante, cuando se ejecute el algoritmo de recomendación.

Otra de las cosas que incluí fue un botón arriba del todo, encima de las rutas que se ofrecen al usuario. La funcionalidad de este botón es mostrar las distintas características de las distintas rutas que se muestran por pantalla. Mi intención es que no se mostrase siempre está información para evitar que sean demasiados los datos que se visualizan a la vez y así no atosigar al usuario:



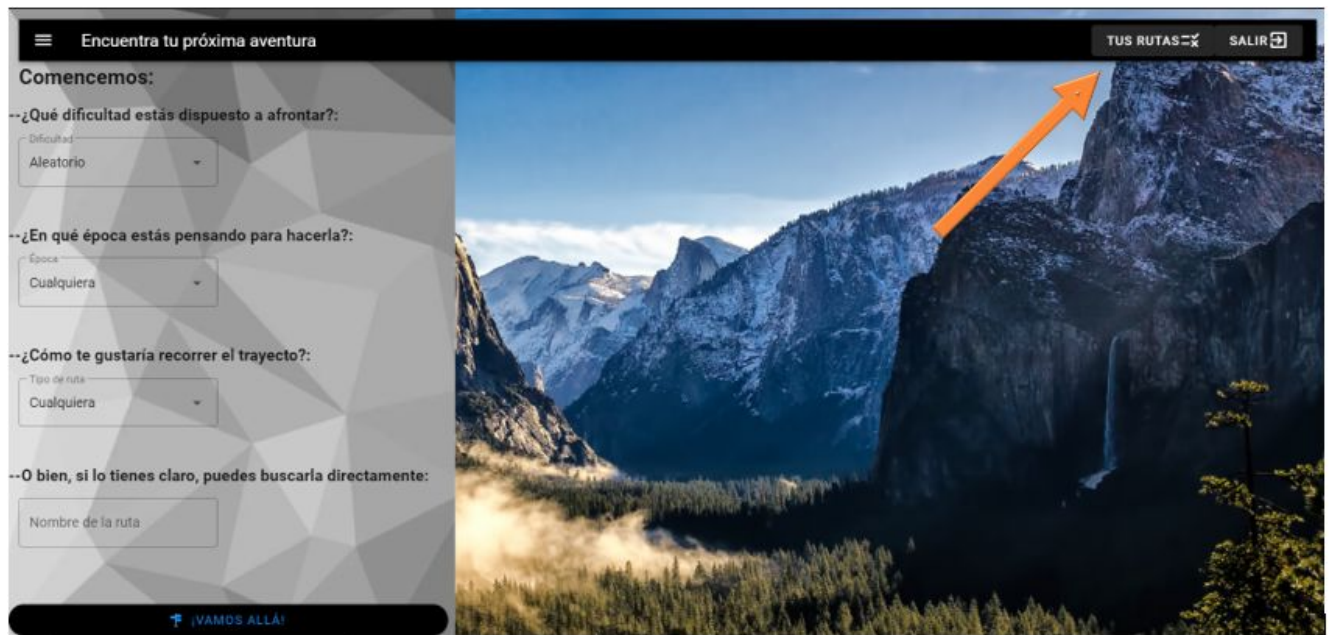
Frontend: Botón no clicado



Frontend: Botón clicado

Por otra parte, dentro de la página principal de cada persona, se dispone de un botón que nos llevará a otra página donde se podrá realizar una visualización de todas las rutas que se hayan realizado y que se hayan marcado como favoritas.

Incluí esta opción como un extra para que todas las personas pudieran recordar que rutas han realizado, ya que si tan solo dispones de un par de rutas o de tres pares de rutas pues quizás no sería necesaria esta sección, pero una vez ya has realizado un número considerable de ellas, es bueno disponer de un apartado dentro de la aplicación para echar un vistazo a todas las aventuras que ha vivido.



Frontend

Una vez se clicca en ese botón, carga otra página donde estarán las rutas favoritas:



Frontend: Rutas favoritas del usuario de ejemplo

La petición que se realiza al backend es similar a la de buscar rutas por el nombre, solo que lo que se le envía al backend es únicamente el nombre del usuario correspondiente y la query que se ejecuta en el backend cambia para buscar las rutas favoritas de ese usuario:

```
axios
.post('http://localhost:3000/listarutas', {
  usuario: this.usuario
})
```

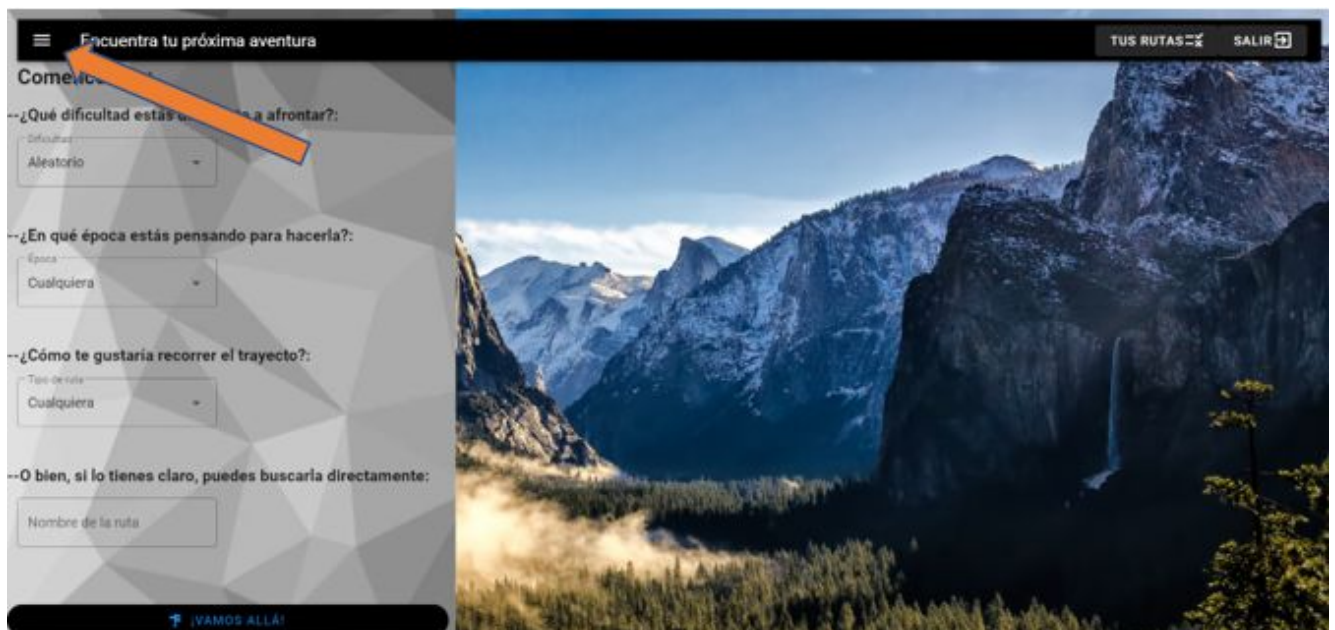
Frontend: Solicitud de rutas favoritas

```
app.post("/listarutas", function (req, res) {
  var usuario = req.body.usuario;
  var rutas = [];
  var query = "MATCH (p:Persona), (r:Ruta)-[:HACER_EN]->(e), (r)-[:DIFICULTAD]->(d), (r)-[:TIPO]->(t) "
    +"WHERE p.usuario='"+usuario+"' AND (p)-[:HA_REALIZADO {experiencia: 'Buena'}]->(r) "
    +"RETURN r.nombre, r.distancia, r.tiempo, r.punto_salida, r.desnivel, r.circular, r.imagen, e.nombre, d.nombre, t.nombre";
```

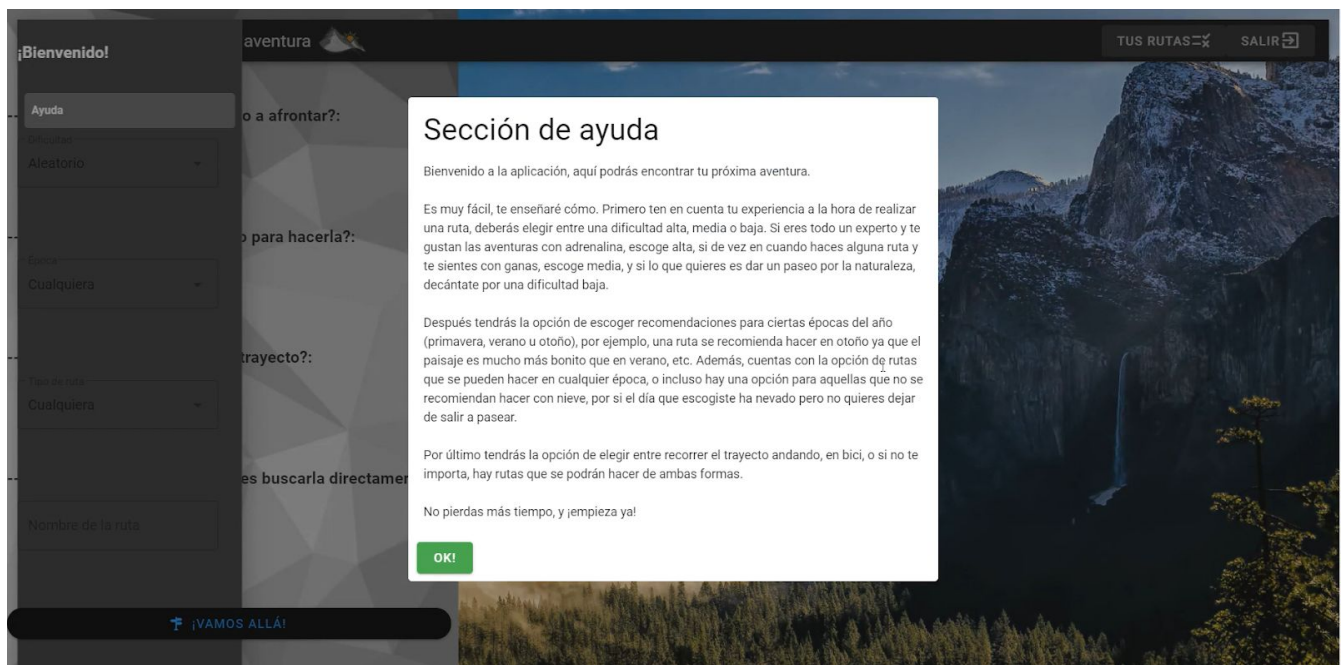
Backend: Obteniendo listado de rutas

Además, el usuario cuenta con una sección de ayuda en la que se le explica brevemente cómo funciona la aplicación. Decidí incluir esta característica ya que quizás hay personas que, cuando utilizan por primera vez esta aplicación, puede que no sepan cómo utilizarla.

En la imagen siguiente vemos donde hay que clicar para abrir el menú donde estará el botón que nos llevará a la ayuda:



Localización de la sección de ayuda



Sección de ayuda

4.Algoritmo de recomendación

4.1.Explicación

Hasta llegar al algoritmo que desarrollé investigué por internet las distintas vías que había de componer el algoritmo de un sistema de recomendación. Para ellos analicé distintas páginas que hablaban sobre las posibilidades que había a la hora de crear el algoritmo pero ninguna me llamaba la atención ya que la mayoría eran demasiado complicados para desarrollarlos en el tiempo que tenía.

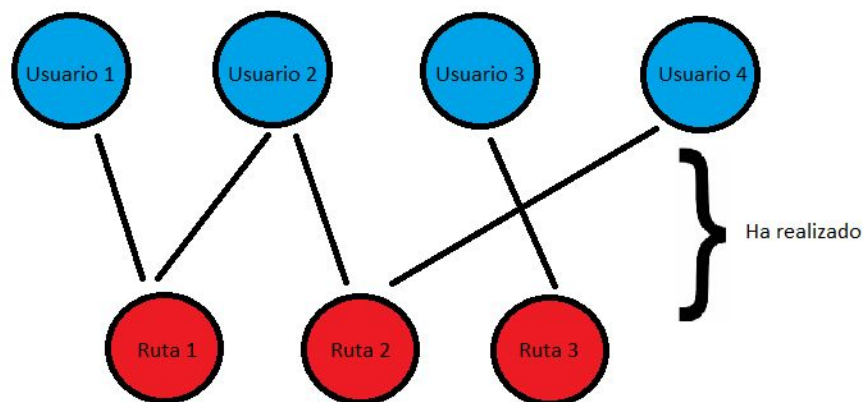
Pero de pronto me tope con un algoritmo denominado Collaborative Filtering el cual se dividía en dos:

- User-based: (Este es el que escogí)
 - Se identifican usuarios similares
 - Se recomiendan nuevos ítems a otros usuarios basado en el rating dado por otros usuarios similares
- Item-based:
 - Calcular la similitud entre ítems
 - Encontrar los “mejores ítems similares” a los que un usuario no tenga evaluados y recomendárselos.

Era genial, user-based era el algoritmo idóneo, ya que en mi aplicación disponía de usuarios y además cada usuario podía marcar como favoritas las rutas que le habían gustado o que le habían parecido más bonitas.

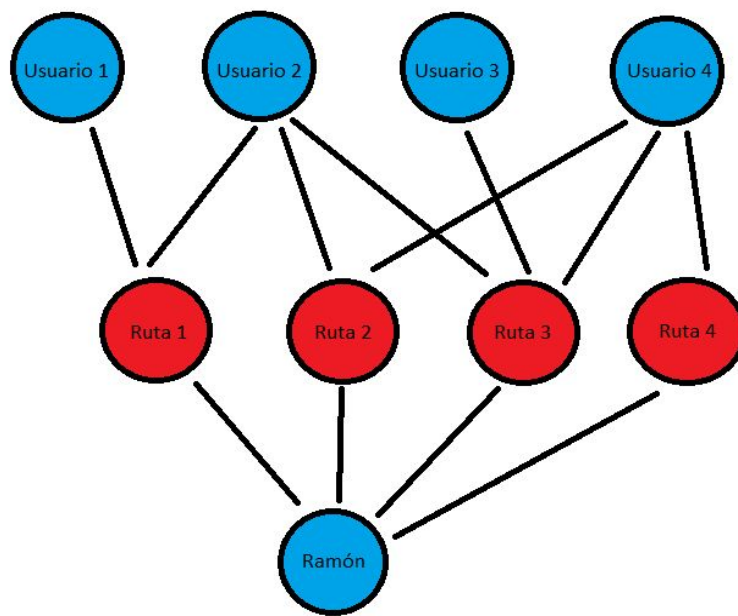
Pero no lo desarrollé exactamente como se explica en algunos libros y en algunas páginas web, ya que aún así me llevaría mucho más tiempo del que en realidad tenía, por lo que decidí hacerlo a mi manera. Y a pesar de hacerlo por mi cuenta el resultado fue muy bueno. Así que sin más dilación voy a explicar el fundamento del algoritmo.

En la base de datos cuento con los distintos usuarios que han estado usando mi aplicación (para poder trabajar con estos usuarios, fueron creados algunos de ejemplo ya que obviamente la aplicación no es pública) y cada uno de estos usuarios cuenta con la opción de añadir rutas a su lista de favoritas.



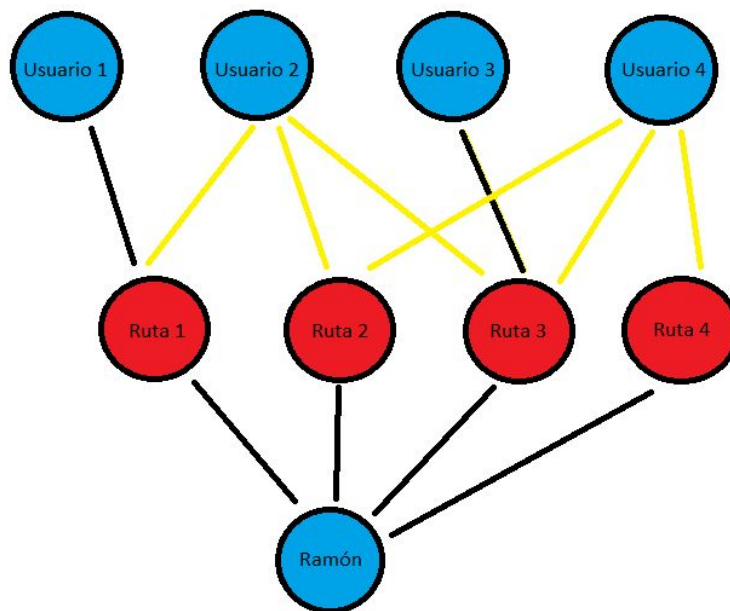
Usuarios y sus rutas

Imaginemos que somos el usuario Ramón y buscamos una ruta con las características X, Y y Z. Primero se buscan las rutas favoritas de este usuario, y después se ve a que otros usuarios les han gustado al menos tres de esas rutas.



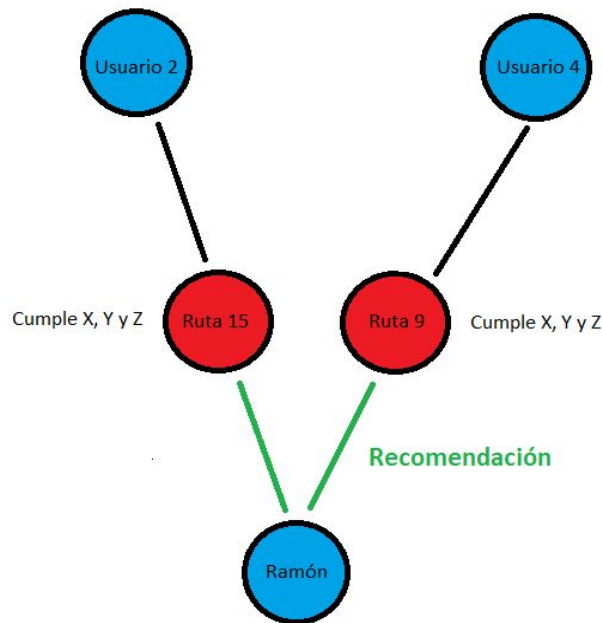
Usuarios con rutas similares a Ramón

En el ejemplo de arriba podemos ver que los usuarios que más coinciden en sus gustos con Ramón son el usuario 2 y el usuario 4, por lo que el algoritmo se centraría ahora en esas dos personas:



Usuarios candidatos

Una vez tenemos a esos usuarios, miramos uno por uno sus rutas favoritas, tratando de encontrar las rutas con las características X, Y y Z que estaba buscando Ramón.

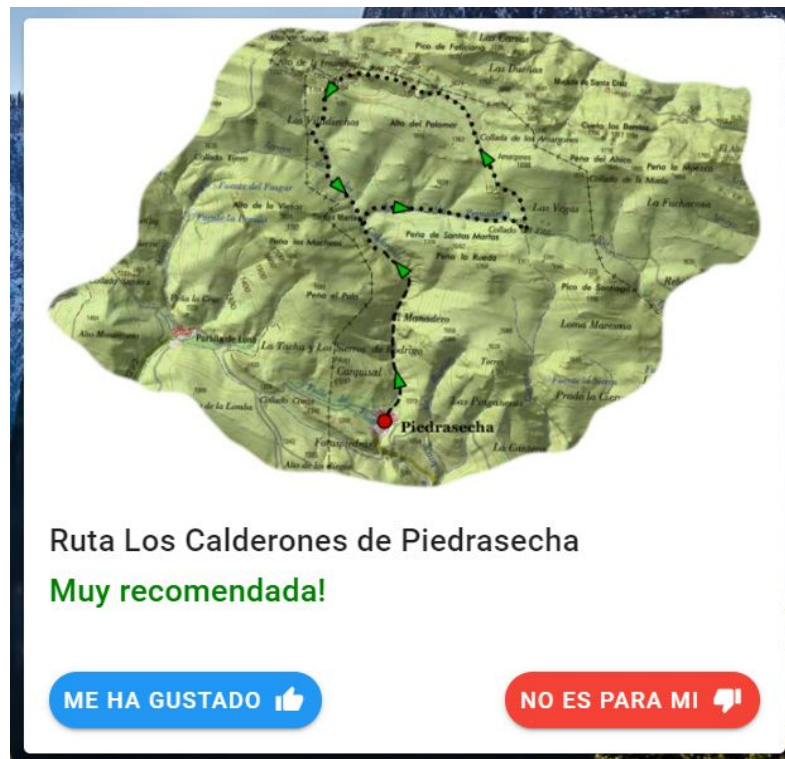


Las rutas recomendables para Ramón según su búsqueda

De esta forma, habrá una probabilidad muy alta de que a Ramón le gusten la “Ruta 15” y la “Ruta 9”, ya que las han realizado usuarios con los mismos gustos, tal y como hemos comprobado en el paso anterior.

En resumen, el sistema analiza qué usuarios coinciden con nosotros en los gustos a la hora de realizar una ruta, y entonces se basa en estos usuarios para ofrecernos una recomendación lo más acertada posible.

Una vez explicado el algoritmo, hablaré sobre cómo lo hice. En primer lugar lo que hice es que se realizara una búsqueda “plana” de las rutas que coinciden con los datos de entrada que aportaba el usuario. Es decir, busco simplemente las rutas que cumplen X, Y y Z (del ejemplo anterior) y una vez las tengo, se aplica el algoritmo. Cuando este finaliza, se marcan las rutas con una etiqueta distintiva (se pone en la descripción de la ruta “Muy recomendable!”) y se colocan en primer lugar, para que el usuario las vea de forma rápida.



Ejemplo de ruta recomendada

Con esto lo que consigo es mostrar más rutas de las que nos proporciona el algoritmo, ya que quizás haya otra ruta que aún no ha realizado nadie y coincide con los gustos del usuario. Así se evita que la aplicación no muestre ninguna ruta cuando en realidad sí que hay disponibles y que además coinciden con lo que está buscando el usuario.

4.2.Código

A continuación veremos parte del código, tanto del backend como del frontend, empleado para el desarrollo del algoritmo.

En primer lugar tenemos la función encargada de realizar una búsqueda teniendo en cuenta simplemente los parámetros de entrada. En la primera imagen se consiguen los datos y se envían a la dirección correspondiente en el backend, la cual ejecutará la query en Neo4j:

```

busqueda_plana: function(){
  console.log('busqueda plana')
  // Make a request
  axios
    .post('http://localhost:3000/busquedaplana', {
      usuario: this.usuario,
      dificultad: this.dificultad,
      epoca: this.epoca,
      tipo: this.tipo
    })
    .then((response) => {
      // handle success
      var json = {msg: 'Error'};

      if(JSON.stringify(response.data)==JSON.stringify(json)){
        alert('No hay rutas que mostrar')
      }else{

        //for(var j=0;j<response.data.length;j++){

          this.rutas = response.data;

        //}
        this.mostrarBotonG = true;
      }
    })
}

```

Frontend

```

app.post("/busquedaplana", function (req, res) {
  var usuario = req.body.usuario;
  var dificultad = req.body.dificultad;
  var epoca = req.body.epoca;
  var tipo = req.body.tipo;
  var rutas = [];
  var query = "MATCH (r:Ruta), (p:Persona), (r)-[:DIFICULTAD]->(d), (r)-[:HACER_EN]->(e), (r)-[:TIPO]->(t) "
    + "WHERE p.usuario = '"+usuario+"' AND d.nombre='"+dificultad+"' AND e.nombre='"+epoca+"' AND t.nombre='"+tipo+"' "
    + "AND NOT EXISTS((p)-[:HA_REALIZADO {experiencia: 'Mala'}]->(r))"
    + "RETURN r.nombre, r.distancia, r.tiempo, r.punto_salida, r.desnivel, r.circular, r.imagen";
}

```

Backend

La siguiente función que veremos será la que obtiene las rutas favoritas del usuario principal, al que llamaré Usuario_1 en adelante, para después poder compararlas con los otros usuarios que coinciden en gustos con él. La llamada se hace a la misma dirección del backend que usamos para rellenar la ventana de rutas favoritas de cada usuario:

```
rellena_rutas_favoritas: function(){
  // Make a request
  axios
    .post('http://localhost:3000/listarutas', {
      usuario: this.usuario
    })
    .then((response) => {
      // handle success
      var json = {msg: 'Error'};

      if(JSON.stringify(response.data)==JSON.stringify(json)){
        console.log('No hay rutas favoritas para este usuario')
      }else{
        this.rutas_fav = response.data;
        console.log(this.rutas_fav)
      }
    })
}
```

Frontend

Ahora, lo que necesitamos es encontrar esos usuarios que coinciden en gustos con el Usuario_1. Para ello se ejecutará la siguiente función:

```
usuarios_rel: function(){
  for(var i=0;i<this.rutas_fav.length;i++){
    // Make a request
    axios
      .post('http://localhost:3000/usuariosrelacionados', {
        ruta: this.rutas_fav[i].nombre_ruta
      })
      .then((response) => {
        // handle success
        var json = {msg: 'Error'};

        if(JSON.stringify(response.data)==JSON.stringify(json)){
          console.log('Ha ocurrido un error')
        }else{
          for(var j=0;j<response.data.length;j++){
            this.usuarios_relacionados.push(response.data[j]+'')
          }
        }
      })
  }
}
```

Frontend

Su principal función es ir a buscar al backend los usuarios que han realizado las mismas rutas que nosotros, ejecutando este proceso para cada una de ellas. Código del backend:

```
app.post("/usuariosrelacionados", function (req, res) {
  var ruta = req.body.ruta;
  var usuarios = [];
  var query = "MATCH (p:Persona), (p)-[:HA_REALIZADO]->(r) "+
    "WHERE r.nombre='"+ruta+"' "+
    "RETURN p.usuario";
```

Backend

Genial, hemos avanzado mucho. Lo siguiente que haré ahora será eliminar los usuarios que se repiten, para evitar un array demasiado largo.

Una vez tenemos los nombres de los usuarios que nos importan en este momento, procedemos a investigar cuáles de sus rutas favoritas coinciden con las especificaciones que está buscando Usuario_1. Para esto, disponemos de la siguiente función:

```
buscar_rutas_recomendadas: function(){
  for(var i=0;i<this.usuarios_rel_def.length;i++){
    // Make a request
    axios
      .post('http://localhost:3000/rutasrecomendadas', {
        usuario: this.usuarios_rel_def[i],
        dificultad: this.dificultad,
        epoca: this.epoca,
        tipo: this.tipo
      })
      .then((response) => {
        // handle success
        var json = {msg: 'Error'};

        if(JSON.stringify(response.data)==JSON.stringify(json)){
          console.log('No hay rutas que mostrar teniendo en cuenta usuarios relacionados')
        }else{

          for(var j=0;j<response.data.length;j++){
            //Comparamos con todas las rutas que ya hay al haber realizado busqueda_plana() antes
            //y si coincide alguna con la recomendacion personal, se le añade el atributo 'Muy recomendada!'
            for(var k=0;k<this.rutas.length;k++){

              if(response.data[j].nombre_ruta == this.rutas[k].nombre_ruta){
                this.rutas[k].recomendada = 'Muy recomendada!';
                break;
              }
            }
          }
          this.mostrarBotonG = true;
        }
      })
  }
}
```

Frontend

```

app.post("/rutasrecomendadas", function (req, res) {
  var usuario = req.body.usuario;
  var dificultad = req.body.dificultad;
  var epoca = req.body.epoca;
  var tipo = req.body.tipo;
  var rutas = [];
  var query = "MATCH (r:Ruta), (p:Persona), (p)-[:HA_REALIZADO {experiencia: 'Buena'}]->(r),
(r)-[:DIFICULTAD]->(d), (r)-[:HACER_EN]->(e), (r)-[:TIPO]->(t) "
    + "WHERE p.usuario = '"+usuario+"' AND d.nombre = '"+dificultad+"' AND e.nombre
= '"+epoca+"' AND t.nombre = '"+tipo+"' "
    + "RETURN r.nombre, r.distancia, r.tiempo, r.punto_salida, r.desnivel,
r.circular, r.imagen";

```

Backend

Y con esto ya tendríamos programado el algoritmo. Una vez acabado procedemos a probar que funciona correctamente, lo cual lo haremos en la sección siguiente.

5.Observación de resultados

A continuación veremos unos ejemplos para observar el correcto funcionamiento de la aplicación y de todo su entramado.

Para empezar tenemos los siguientes usuarios con las correspondientes rutas como favoritas:

Miguel ->	Faedo -> me gusta
	Calderones -> me gusta
	Camino ancares -> me gusta
	Pinares del Manzanal -> no me gusta
Fran ->	Faedo -> me gusta
	Calderones -> me gusta
	Camino ancares -> me gusta
	Pluma -> me gusta
Rio lobos	(rio_lobos)-[:HACER_EN]->(siempre), (rio_lobos)-[:DIFICULTAD]->(media), (rio_lobos)-[:TIPO]->(bici),
	-> me gusta

Gus -> Faedo -> me gusta

Pinares del Manzanal -> me gusta

Moncayo -> me gusta

Lago de Sanabria → (sanabria_lago)-[:HACER_EN]->(otoño),
 (sanabria_lago)-[:DIFICULTAD]->(media),
 (sanabria_lago)-[:TIPO]->(bici), -> me gusta

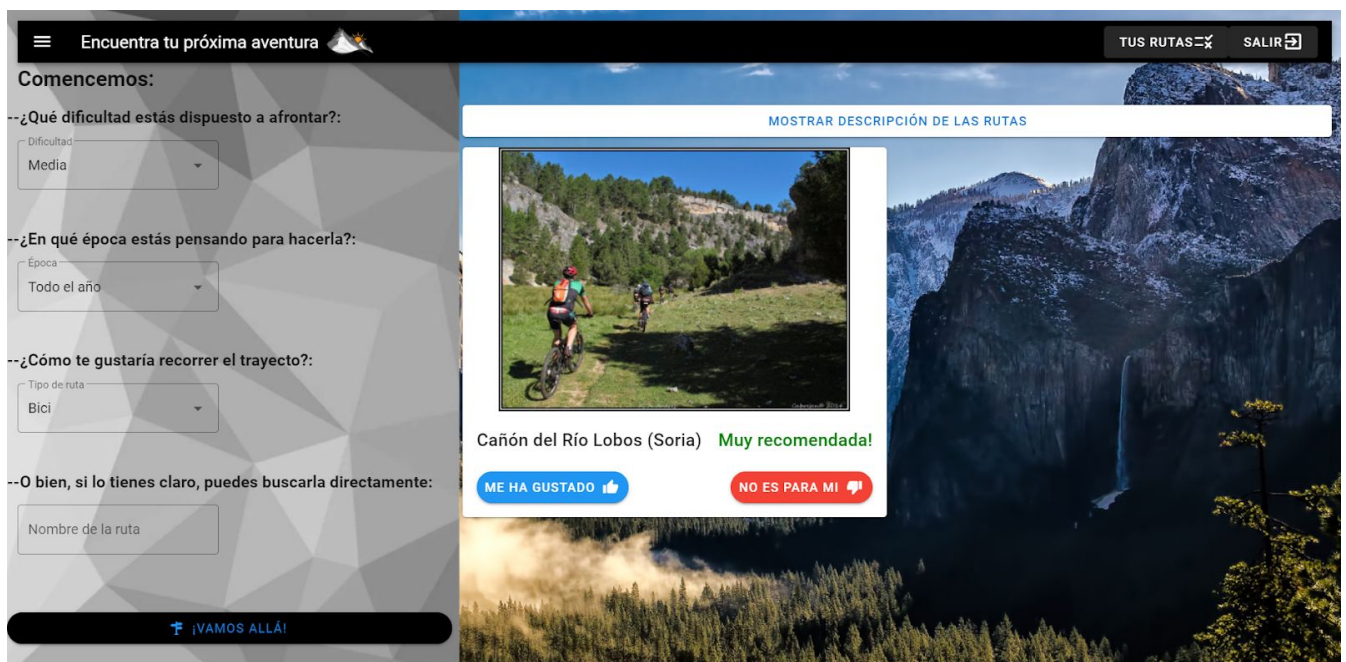
Mampodre → (mampodre)-[:HACER_EN]->(siempre_no_nieve),
 (mampodre)-[:DIFICULTAD]->(media),
 (mampodre)-[:TIPO]->(andando), -> me gusta

Pepe -> Pinares del Manzanal -> me gusta

Moncayo -> me gusta

Faedo -> me gusta

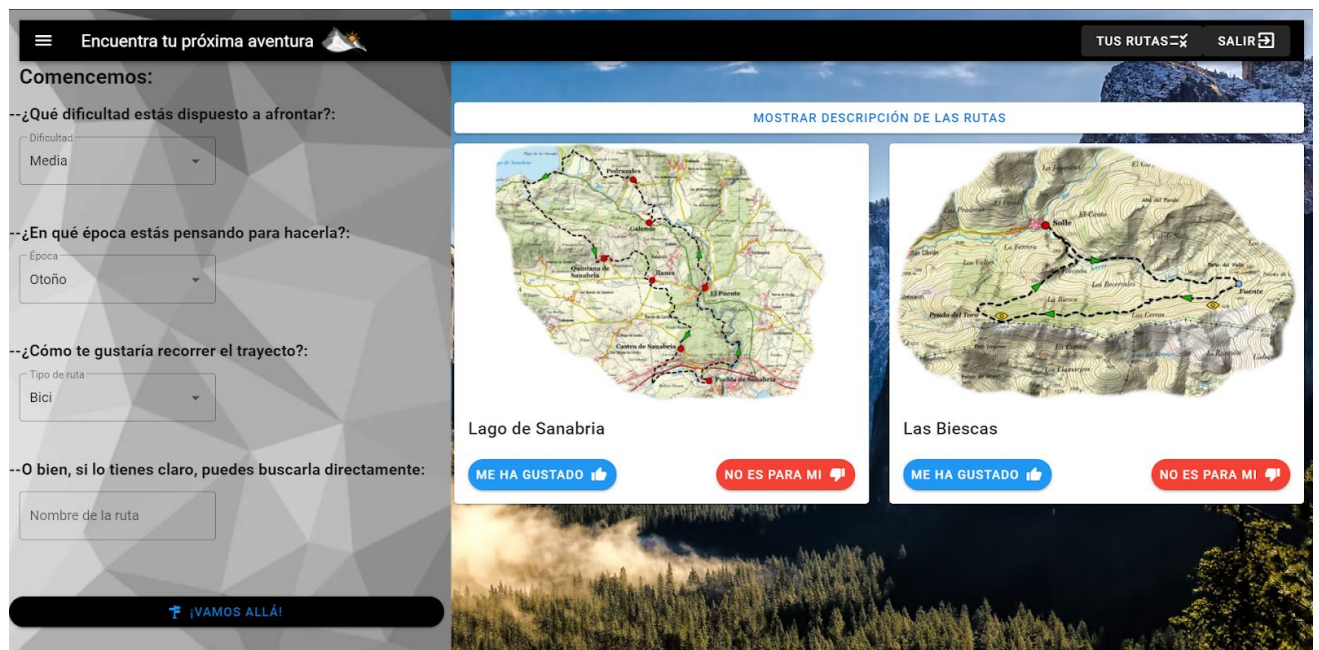
Como podemos observar, el usuario Miguel coincide en 3 rutas con Fran. Por lo tanto, si buscamos una ruta con las características de poder realizarse todo el año, con dificultad media y destinada a hacerse en bici, el sistema nos tendrá que marcar como “Muy recomendada” la ruta de Río lobos. Veámoslo:



Ejemplo de recomendación

Efectivamente, esa es la que nos recomienda. No aparecen más rutas debido a que dentro de la base de datos no dispongo de más rutas con esas mismas características, ya que la base de datos, es pequeña aún.

Por otro lado vemos que este usuario Miguel, solo coincide en 1 ruta con Gus, pero el requisito indispensable para que un usuario sea candidato para llevar a cabo la recomendación es que le hayan gustado las 3 mismas rutas o más. Por lo que si buscamos una ruta con dificultad media, recomendada para hacer en el otoño y que se realice en bici, nos mostrará el lago de Sanabria, pero no nos lo marcará como “Muy recomendada”:

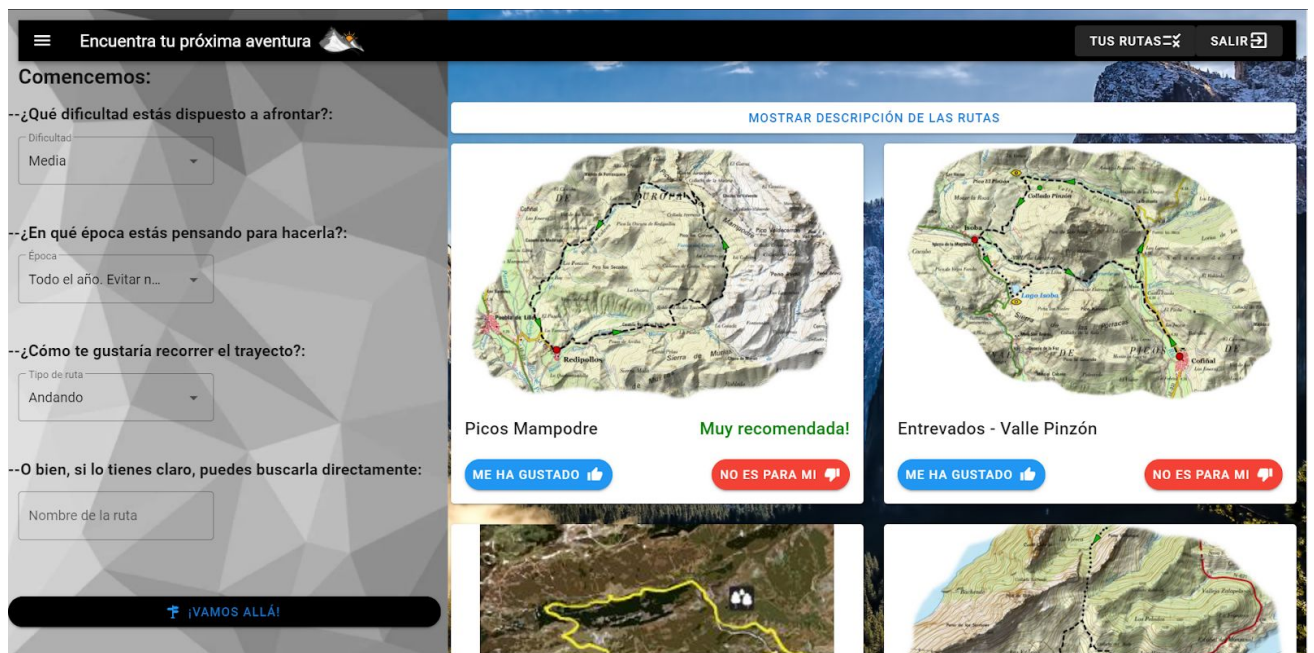


Ejemplo de recomendación

Y como podemos ver en la imagen de arriba, nos recomienda esa ruta pero sin marcarla, ya que no se cumplen las condiciones del algoritmo.

Vamos a probar otro ejemplo para asegurarnos de que funciona correctamente.

Seremos ahora el usuario Pepe, el cual coincide en 3 rutas con Gus (Pinares del Manzanal, Faedo y Moncayo). Si, desde la cuenta de Pepe, buscamos una ruta que se haga andando, de dificultad media y que la época sea cualquiera, pero evitando épocas de nieve, nos debería de recomendar la ruta de picos Mampodre. Veámoslo:



Ejemplo de recomendación

Perfecto, es la primera ruta que nos muestra. Y con estos ejemplos hemos comprobado que funciona bien la aplicación y que el algoritmo está bien programado.

Si fuera una aplicación real, en el sentido de que la gente la pudiera usar públicamente, el algoritmo sería aún más preciso, ya que habría muchas más rutas almacenadas y muchas más personas registradas en la base de datos.

6.Análisis

Cuando empecé con este trabajo no tenía ni idea de cómo afrontarlo. Sabía que necesitaba utilizar Neo4j para crear la base de datos, pero era una herramienta totalmente nueva para mi y para todos mis compañeros, por lo que ya solo acostumbrarme al entorno me iba a llevar unas cuantas horas. Por otro lado estaba Cypher, el lenguaje de programación de Neo4j, también nuevo para mi, por lo que tuve que buscar por internet mucha información para entender su funcionamiento y aprender las nociones básicas.

En primer lugar empecé a investigar sobre Neo4j, y para ello empecé a ver los videos de YouTube que nos recomendó Enrique, mi profesor en esta asignatura. En seguida comprendí cómo funcionaba esta nueva plataforma y por qué había cobrado tanta importancia en los últimos años. Lo que me costó un poco más fue Cypher. A día de hoy como informáticos hemos lidiado con muchos lenguajes de programación, pero siempre cuesta empezar a aprender uno nuevo. Sobre todo con lo que más aprendí sobre este lenguaje y sobre Neo4j fue gracias al curso de esta

tecnología que me hice en la propia página, donde se tocan todos los temas de una forma muy amplia. Estoy bastante contento del trabajo de investigación que lleve a cabo.

Una vez tenía los conocimientos, tocaba empezar a trabajar en el tema, y como ya he comentado no fue fácil llevarlo a cabo. Era una temática que me gustaba mucho, pero no tenía nada para empezar, así que empecé de cero. Con lo que había aprendido en el curso y por internet cree mi propia base de datos de la cual estoy muy orgulloso, a pesar de haberme llevado bastantes horas de búsqueda de información y de programación. Cuando vi el resultado final todo mereció la pena, obviamente no disponía de mil rutas ni de trescientas, pero había suficientes para reflejar mis objetivos y para desarrollar cómodamente el proyecto.

Durante el desarrollo de la aplicación me lo pasé muy bien, ya que estaba haciendo algo 100% mío y de un tema que me gustaba mucho. No me costaba ponerme a programar, aunque luego me topaba con algún que otro problema, ya que siempre los acaba solucionando a base de hacer pruebas y analizar el código.

Considero que el resultado no está mal para el tiempo en el que lo he desarrollado, aunque soy consciente de que se podría haber hecho mejor. Estoy muy contento con la aplicación que he logrado.

6.1.DAFO

- **Debilidades** - La cantidad de información de la que dispone la base de datos es poca en comparación con las grandes bbdd que encontramos en kaggle o con las que cuentan otras muchas páginas web. A la hora de ir a buscar una ruta con ciertos parámetros, es posible que no se disponga de ninguna para mostrar, o que se muestren dos o tres, y una vez el usuario ya haya realizado esas dos o tres rutas, la aplicación no le va a recomendar nada nuevo que no haya visto antes.
- **Amenazas** - La principal amenaza a la que me he enfrentado es a la falta de documentación para crear el tipo de aplicación que tenía que hacer, así como que lo más importante que tenía que utilizar eran herramientas nuevas, como Neo4j y Cypher. Debido a esto es por lo que, antes de empezar a desarrollar el proyecto, tuve que buscar mucha información para primero saber a qué me enfrentaba y como podía llevarlo a cabo.
- **Fortalezas** - El sistema que cree de usuarios y el algoritmo que elegí desarrollar son muy completos. Con los usuarios hago que cada persona tenga su rincón personal dentro de la aplicación para que pueda tener guardadas las rutas que hace todas en un mismo sitio. En lo que respecta al algoritmo los usuarios son el centro de su lógica, y apoyándome en ellos se

pueden ofrecer recomendaciones muy buenas y personalizadas para cada tipo de usuario.

- **Oportunidades** - La verdad que el abanico de oportunidades que hay es muy amplio, ya que no hay ninguna empresa ni ningún negocio que haya explotado esto como yo lo planteo. No hay ningún sitio que te recomiende rutas de una forma tan personalizada y atendiendo a tus experiencias anteriores. Por lo que las oportunidades que se brindan son muchas, desde ampliar la base de datos introduciendo más información trabajando con un grupo de varias personas, hasta realizar una mejora del algoritmo para que sea más profesional o más acertado introduciendo más parámetros a la hora de buscar una nueva ruta.

7. Líneas de futuro

De cara al futuro cuento con varias posibilidades y algunos nichos de mercado sin explotar. Si consultamos a Google cuales son las rutas que deberíamos hacer, nos ofrece resultados basados en opiniones globales de la persona o personas que han escrito las distintas páginas web, no es una recomendación personal para el usuario que está buscando la información.

Es por esto que mi concepto cuenta con una gran ventaja respecto a este sistema, como ya he estado comentando a lo largo del proyecto, mi aplicación realiza recomendaciones en función de los usuarios más afines a nosotros, es decir, con los que más gustos en común tenemos, por lo que las rutas ofrecidas encajan bastante con nuestros deseos.

Por eso, pensando en el futuro solo se me ocurren mejoras que hagan crecer más y más esta idea. De las primeras tareas que hay por delante es llevar a cabo una ampliación de la base de datos (ya que con la que contamos ahora la he realizado yo a mano en un plazo de tiempo muy justo), incluyendo cientos y cientos de rutas más. El objetivo de esto es hacer que la bandeja de posibilidades sea cada vez más grande, ofreciendo al usuario, por ejemplo, cuarenta rutas entre las que elegir en vez de cuatro o cinco, que es con lo que contamos ahora mismo.

Otro de los frentes que tengo abiertos es ampliar el área en el que se centra la aplicación. Sobre todo gira en torno a rutas de senderismo que se pueden realizar en León, alrededores o en Castilla y León en general. El objetivo sería incluir rutas de toda España, desde Galicia o Asturias hasta Sevilla o Málaga. Con toda esta información la aplicación no solo servirá para Leoneses, sino para gente de toda España, donde si por ejemplo te encuentras en Valencia, y no sabes que hacer un domingo, puedas ver qué recomendaciones tienes para hacer una escapada.

Por otro lado, otra de las cosas que me gustaría implementar, si en algún momento continuo con este proyecto, sería el tema de la seguridad. Con esto me refiero a tener la base de datos en un lugar seguro y que nadie pueda acceder a ella, ya que de esta forma se podría descargar información privada de las cuentas de los usuarios, y es algo que nunca debería pasar.

Por último, me gustaría incluir que para registrarse sea necesario introducir el correo electrónico, de esta forma se podrían enviar correos a los usuarios cuando lleven un tiempo sin usar la aplicación, invitándoles a buscar su próxima aventura.

8. Lecciones aprendidas

Para poder llevar a un buen puerto este proyecto y que el resultado fuera lo que yo esperaba, he tenido que trabajar mucho. He aprendido que sea cual sea el reto que me pongan por delante seré capaz de realizarlo a base de dedicarle mucho tiempo y esfuerzo.

Como ya he comentado, he hecho una labor de investigación antes de empezar a hacer el trabajo, ya que muchas herramientas de las que he utilizado eran nuevas para mí. Sobre todo Neo4j y todo lo que le rodea. Pero he sabido dividir el tiempo correctamente para aprender sobre esta tecnología a base de cursos, videos en internet, artículos, etc.

En definitiva, he realizado un proceso de investigación amplio antes de empezar a programar nada, y gracias a esto me he dado cuenta de que cuando te ponen un reto delante y es algo nuevo para ti, no tienes que empezar a trabajar en él de inmediato y a lo loco, sino que hay una etapa de aprendizaje previa, antes de entrar en el meollo de la cuestión. Esto es muy importante, ya que gracias a esta fase de búsqueda de información, pude crear los cimientos para que el trabajo que venía a continuación se desarrollara más rápido, ya que como he dicho, tendremos los conocimientos necesarios y sabremos cómo proceder en el desarrollo.

9. Referencias y bibliografía

9.1. Referencias

1. https://neo4j.com/graph-algorithms-book-b/?utm_expId=.kpRQ-jdpTc6d0jinTB4Jag.1&utm_referrer=
2. <https://neo4j.com/developer/language-guides/>
3. <https://github.com/neo4j/neo4j-javascript-driver>
4. <https://github.com/neo4j/neo4j-javascript-driver/blob/4.2/examples/node.js>
5. <https://vuejs.org/>
6. <https://vuetifyjs.com/en/>
7. <https://developer.mozilla.org/es/docs/Web/JavaScript>
8. <https://es.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html>
9. <https://www.electronjs.org/>
10. <https://nodejs.org/es/>
11. <https://expressjs.com/es/>
12. <https://developer.mozilla.org/es/docs/Web/JavaScript>
13. <https://neo4j.com/developer/language-guides/>
14. <https://neo4j.com/>
15. <https://neo4j.com/developer/cypher/>
16. <https://code.visualstudio.com/>
17. <https://www.kaggle.com/>

9.2. Bibliografía

1. Artículo sobre sistemas de recomendación:
<https://medium.com/jaywrkr-tech/gu%C3%ADa-para-construir-un-sistema-de-recomendaci%C3%B3n-parte-1-2b1a65d6eac3>
2. Algunos videos de Neo4j:
 - a. https://www.youtube.com/watch?v=BcgXw06ISlo&feature=emb_title&ab_channel=Neo4j
 - b. https://www.youtube.com/watch?v=5TI8WcaqZoc&list=WL&index=2&t=11s&ab_channel=Neo4j
 - c. https://www.youtube.com/watch?v=-dCeFEqDkUI&t=436s&ab_channel=Neo4j
3. Algunos videos sobre Cypher:
 - d. https://www.youtube.com/watch?v=NH6WoJHN4UA&t=1s&ab_channel=Neo4j
 - e. https://www.youtube.com/watch?v=NO3C-CWykkY&t=686s&ab_channel=Neo4j
4. Enlace al curso de Neo4j y Cypher:
<https://neo4j.com/graphacademy/training-intro-40/enrollment/>