



universidad
de león



Escuela de Ingenierías
Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Sistemas de Información y Gestión
de Business Intelligence



Sistema de Recomendación de Tapas y Bares

Autor:

Raúl Seara Barroso

Tutor:

Enrique López González

Curso 2020-2021

Índice

1.	Introducción	2
2.	Descripción del problema	3
3.	Herramientas utilizadas	4
i.	Neo4j.....	4
ii.	OSF	4
iii.	GitHub.....	5
iv.	Visual Studio Code.....	5
v.	Node.js.....	6
vi.	Vue	6
vii.	Vuetify	7
viii.	CorelDraw x7	7
4.	Descripción de la aplicación	8
i.	Base de Datos	9
ii.	Backend y Frontend.....	13
➤	Login/Registro	14
➤	Pantalla de Inicio	18
➤	Buscar Por Tapa	27
➤	Bares	33
➤	Búsqueda Personalizada	35
5.	Algoritmo empleado – Voy a tener TapO’N	38
6.	Análisis de los resultados	42
➤	Primer caso:	42
➤	Segundo caso:.....	45
7.	DAFO	49
8.	Líneas de futuro.....	50
9.	Lecciones aprendidas	51
10.	Agradecimientos.....	52
11.	Bibliografía y referencias	53

1. Introducción

En el momento de escribir esta memoria, el mundo vive un momento de oscuridad. Quizás uno de tantos a lo largo de la historia, pero distinto porque ahora parecía que nos habíamos acostumbrado a vivir sin guerras, relativamente cómodos, sin miedo a despertar un día y que todo fuera distinto... Es verdad que todo esto es muy general, porque no todo el mundo disponía de esas comodidades, pero incluso quien menos tenía podía salir a la calle y reunirse con sus seres queridos. Nos han privado del mayor tesoro que teníamos y que no pensábamos perder: la libertad. No podemos reunirnos con quien queramos, no podemos salir a determinadas horas, hay muchísimos comercios cerrados y personas que no pueden trabajar... Nos hemos convertido en nuestros perros, esperando a que nos dejen ir a pasear y cumpliendo las normas. Todos juntos, ahora que eso de pensar en uno mismo estaba tan de moda.

Es paradójico porque esta aplicación consiste en un sistema de recomendación de tapas y bares y me resulta triste pensar que ahora mismo ninguno de los bares que están en ella pueden abrir. Nadie puede ir a degustar sus comidas ni a tomarse algo con sus amigos. Pero si algo nos ha enseñado la historia es que estamos condenados a superarnos. Saldremos adelante y volveremos a estar todos juntos celebrando que la vida sigue y el sol sale de nuevo. Volveremos a nuestros bares, volveremos porque *“el hombre es un ser social por naturaleza”* como dijo Aristóteles. Un paso atrás puede convertirse en siete hacia delante si cogemos lo que hemos aprendido y disfrutamos cada día como si fuese el último, porque nos equivocábamos: un día te despiertas y todo ha cambiado. Nada es para siempre, aunque eso sí lo sabíamos, ¿o no?

Toca disfrutar cada tapa como si fuera la última. TapO'N.

2. Descripción del problema

Al igual que León, la ciudad en la que se basa este sistema de recomendación, existen muchas ciudades donde hay bares que ofrecen tapas, un producto que, a día de hoy, es tan importante en el mundo culinario como un buen plato elaborado por un reconocido chef. Las tapas son un arte y están ganando cada día más reconocimiento, tanto que son consideradas alta cocina en tamaño mini. Tomarse una caña (por ejemplo) acompañada de una buena tapa, era y es un placer para el paladar y los clientes saben reconocer qué bar es mejor ya no solo por los precios o por la calidad de su atención al público, si no por las tapas que hacen.

Habiendo tantos, sin saber la carta de cada uno, en muchos casos, muy difícil de encontrar online porque a veces no tienen sitio web y se transmiten por el boca a boca (algo que he comprobado de primera mano buscando información), es complicado para quienes sean turistas o no conozcan bien su ciudad acertar a la primera con un bar donde les vayan a ofrecer una tapa que sea de su gusto.

Para ayudar en esta búsqueda nace TapO'N, mi aplicación web, que es un sistema de recomendación basado en encontrar bares a partir de la selección de tapas. Pero no se queda solo en eso, porque también ofrece la posibilidad de añadir a esas búsquedas parámetros como que los usuarios puedan llevar a su perro, tengan canal de fútbol o cerveza artesana, entre otros, incrementando las posibilidades de tener una gran experiencia en el establecimiento que escojan.

En esta aplicación me he centrado en los principales bares del Barrio Húmedo de León, buscando información sobre sus tapas y cogiendo una enorme variedad de ellas para que los usuarios tengan dónde elegir. También he buscado las posibilidades que ofrecían a sus clientes como complementos.

Tanto las funciones que ofrece TapO'N como su base de datos están detalladas en el punto 4. Descripción de la Aplicación.

3. Herramientas utilizadas

i. Neo4j



Ilustración 1 - Logo de Neo4j

Según su propia página, “Neo4j es una base de datos de gráficos nativa, construida desde cero para aprovechar no solo los datos sino también las relaciones de datos. Neo4j conecta los datos a medida que se almacenan, lo que permite realizar consultas nunca antes imaginadas, a velocidades que nunca se creyeron posibles.” (Neo4j, 2020)

Neo4j no es una simple base de datos, es mucho más que eso. Es la pieza fundamental que sostiene a la aplicación. Al principio me tuve que hacer a la idea de que no era como las que había usado, estrictas, con tablas y claves foráneas. La flexibilidad de Neo4j y su fácil aprendizaje, a pesar de que me llevó muchas horas aprender a dominar Cypher (el lenguaje que emplea en las consultas), son la clave del funcionamiento de la aplicación. Al poder crear relaciones entre sus nodos y borrar con facilidad, sumado a que puede devolver datos de distintos tipos de nodos agrupados, buscar por propiedades... Todo contribuyó y las horas empleadas dieron sus frutos, porque no solo me certifiqué como profesional en la herramienta gracias a su propia página oficial, si no que a la hora de programar el backend de TapO'N fue relativamente sencillo para mí crear las consultas necesarias.

La versión de Neo4j que utilicé fue la de escritorio porque quería una base a largo plazo para hacer todas las pruebas, pero tiene más versiones, como la Sandbox para hacer bases temporales.

ii. OSF



Ilustración 2 - Logo de OSF

“OSF es una plataforma abierta y gratuita para respaldar su investigación y permitir la colaboración.” (OSF, 2020)

Es imposible hacer esta memoria sin hacer una mención a otra de las herramientas con las que he desarrollado el proyecto. El OSF ha sido el cuaderno de trabajo en el que he ido anotando cada uno de los pasos que daba mientras iba evolucionando el desarrollo y mientras yo iba aprendiendo conceptos y teniendo ideas. Al principio me resultó extraño, porque siempre me ha gustado el papel, pero una vez me hice con su

funcionamiento me resultó muy útil porque realmente es cómodo poder guardar todo en un mismo lugar y además permite conectarse con GitHub, una herramienta fundamental que explicaré ahora.

iii. GitHub



Ilustración 3 - Logo de GitHub

“GitHub es una plataforma de alojamiento de código para el control de versiones y la colaboración. Le permite a usted y a otros trabajar juntos en proyectos desde cualquier lugar.” (GitHub, 2020)

Esta herramienta es una de las que más he utilizado a lo largo de la carrera y aunque su propósito es simple, pues consiste en guardar versiones de los proyectos de desarrollo de software, es muy eficaz en su propósito y me gusta mucho. A pesar de haber trabajado solo, la he utilizado para subir mi código mientras lo desarrollaba porque me resulta muy cómodo y me gusta saber que mi proyecto está seguro al tenerlo localmente y también almacenado en la nube.

iv. Visual Studio Code

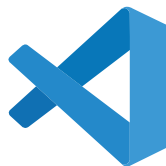


Ilustración 4 - Logo de Visual Studio Code

“Visual Studio Code es un editor de código fuente ligero pero potente que se ejecuta en su escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C ++, C #, Java, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity).” (Visual Studio Code, 2020)

Esta herramienta es muy muy útil para programar. He usado otros editores, Eclipse sobre todo, pero VSC me encanta. Lo he utilizado para desarrollar el código tanto del back como del front y además de muchísimos plugins que ofrece, tiene conexión directa con GitHub para añadir los cambios y eso para hacer los proyectos en equipo durante la carrera me ha servido mucho y me ha ahorrado tiempo.

v. Node.js



Ilustración 5 - Logo de Node.js

“Node.js es un entorno de ejecución de JavaScript de código abierto, multiplataforma. Ejecuta código JavaScript fuera de un navegador.” (Node.js, 2020)

Esta herramienta permite recibir en el backend las consultas que hace el cliente desde el frontend, las procesa y permite responder a esas peticiones. Es decir, trabaja conjuntamente con la conexión a la base de Neo4j. El cliente desde la interfaz hace peticiones que procesa el backend, quien establece conexión con la base, recoge las respuestas de Neo4j y luego las devuelve a la interfaz para que las pueda mostrar al usuario. Es una herramienta fundamental para hacer aplicaciones web y la había utilizado aunque no demasiado.

vi. Vue



Ilustración 6 - Logo de Vue

“Vue (pronunciado / vju: /, como view) es un marco progresivo para construir interfaces de usuario. A diferencia de otros marcos monolíticos, Vue está diseñado desde cero para ser adoptable de forma incremental. La biblioteca principal se centra solo en la capa de vista y es fácil de recoger e integrar con otras bibliotecas o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de impulsar aplicaciones sofisticadas de una sola página cuando se usa en combinación con herramientas modernas y bibliotecas de soporte.” (Vue.js, 2020)

Esta herramienta la he utilizado para construir la interfaz, combinada con la siguiente que es Vuetify. Junto a ella en el frontend, he usado módulos como **Axios** y **Express** para hacer las consultas hacia el backend y **Electron** para que al ejecutar aparezca como si fuera una aplicación de escritorio. Al igual que Node.js, es una herramienta muy utilizada para desarrollar aplicaciones web.

vii. Vuetify



Ilustración 7 - Logo de Vuetify

“Vuetify es un marco de interfaz de usuario completo construido sobre Vue.js. El objetivo del proyecto es proporcionar a los desarrolladores las herramientas que necesitan para crear experiencias de usuario enriquecedoras y atractivas. A diferencia de otros marcos, Vuetify está diseñado desde cero para ser fácil de aprender y gratificante de dominar con cientos de componentes cuidadosamente elaborados a partir de la especificación de Material Design.” (Vuetify, 2020)

Vuetify es un complemento que funciona sobre Vue y permite diseñar interfaces de una forma sencilla y colorida. Me parece relativamente sencillo de utilizar y los resultados son muy interesantes incluso cuando eres novato, aunque a pesar de haberle dedicado muchas horas en este y otro proyecto aún no lo domino del todo.

De cara a los usuarios me parece que sirve para crear aplicaciones muy vistosas y todo lo que se ve, como en la cocina con los platos, se puede comer o en este caso, utilizar, porque es posible dar funcionalidad a todo literalmente.

viii. CorelDraw x7

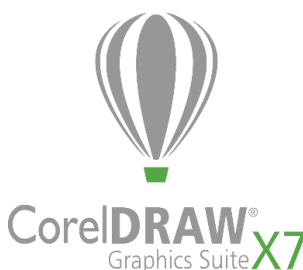


Ilustración 8 - Logo de CorelDraw X7

CorelDraw es un programa que permite hacer diseños vectoriales. Es muy potente, de pago y lo utilizan los diseñadores gráficos. Pude utilizarlo para diseñar el logo de TapO'N gracias a que mis padres lo tienen comprado (puesto que lo utilizan en su trabajo).

Para mí el logo de la aplicación es muy importante, es su señal de identidad y cuando piensas en cualquiera, lo primero que se te viene a la mente es su logotipo, así que me esforcé (y pedí mucho consejo) porque quería que fuese algo bonito y reconocible y espero haberlo conseguido.

4. Descripción de la aplicación

Lo primero que hay que preguntarle a alguien que no conoces es su nombre, por tanto, antes de mostrar quién es TapO'N y cómo funciona, explicaré las raíces del nacimiento de su nombre. Esta aplicación web se basa en tres pilares fundamentales: Neo4j, como está explicado en la sección anterior, las Tapas y por supuesto, León, donde sucede la magia. Tap viene de tapa, y O'N es un juego de palabras entre León y Neo4j, donde el apóstrofe representa la tilde en la O que no existe en inglés y que le da un toque internacional dado que la ciudad es muy turística.

La aplicación está estructurada de la siguiente manera:

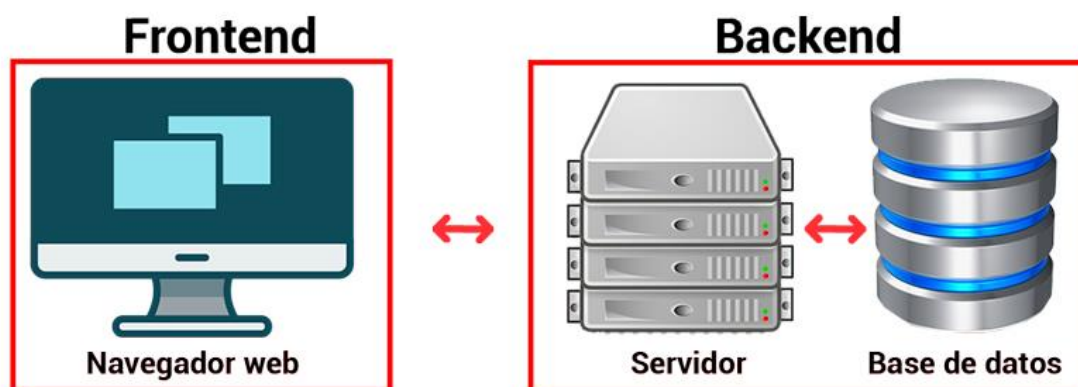


Ilustración 9 – Estructura interna de una aplicación web

- Tenemos por un lado la Base de Datos en Neo4j, donde están almacenados los datos que necesitamos.
- Por otro lado, tenemos el Backend, que es el encargado de recibir las consultas por parte de la interfaz, las procesa y luego realiza peticiones a la base para obtener los datos que quiere el usuario. Una vez los obtiene, los devuelve al front. Se conecta a la base mediante el driver de Neo4j. El usuario no ve esta parte por lo que no es consciente de lo que hace, pero es aquí donde de verdad necesitamos máxima fiabilidad y una buena velocidad de respuesta para que la interfaz pueda satisfacer al usuario con lo que le pida.
- Por último, el Frontend o la parte gráfica, que se encarga de interactuar con el usuario y en función de lo que necesite, realiza las consultas al back. Es lo que el usuario ve y por tanto es una parte delicada porque es necesario que sea atractiva visualmente y al mismo tiempo, fácil de usar, intuitiva y rápida.

Una vez hecha la presentación, podemos empezar a conocer cada una de sus partes con detalle.

i. Base de Datos

La base de datos de TapO'N la construí a mano línea a línea, buscando información sobre bares del barrio húmedo. Encontré una web muy interesante que tenía muchos datos que me hacían falta ([1](#)). Añadí los bares, sus tapas y además creé unos usuarios básicos para poder acceder cuando no estaba hecho el registro, el cual añadí posteriormente para que cualquiera pueda registrarse. He aquí un trozo del documento que crea la propia base.

```
//Nodos de los usuarios
CREATE (paco:Person{name:"Paco Gonzalez", user:"Paco", password:"Paco123"}),
(manolo:Person{name:"Manolo Lama", user:"Manolo", password:"Manolo123"}),
(juanma:Person{name:"Juanma Castaño", user:"Juanma", password:"Juanma123"}),
(roncero:Person{name:"Tomas Roncero", user:"Roncero", password:"Roncero123"}),
(marcos:Person{name:"Marcos Lopez", user:"Marcos", password:"Marcos123"}),
(manu:Person{name:"Manu Carreño", user:"Manu", password:"Manu123"})

//Nodos de los bares, debajo de cada uno sus tapas
CREATE (caferua:Bar {name:"Cafe Bar La Rua", address: "Calle la Rúa, 11, 24003 León", telephone:"+34 987 17 86 37", web:"rua11.es"}),
(anchoa:Tapa {tipoTapa:"Anchoa"}),
(boqeron:Tapa {tipoTapa:"Boqeron"}),
(queso:Tapa {tipoTapa:"Queso"}),
(sidarta:Bar {name:"Sidarta", address: "Calle Cascaleria, 3, 24003 León", telephone:"+34 987 04 04 53", web:"gastrolateria-sidarta.neg"},
(nachos:Tapa {tipoTapa:"Nachos"}),
(embutido:Tapa {tipoTapa:"Embutido"}),
(looking:Bar {name:"Looking For", address: "Rincón Conde Rebolledo, 13, 24003 León", telephone:"+34 987 03 28 41", perros: "si"}),
(rana:Tapa {tipoTapa:"Ancas de rana"}),
(caracoles:Tapa {tipoTapa:"Caracoles"}),
(empanada:Tapa {tipoTapa:"Empanada"}),
(sepia:Tapa {tipoTapa:"Sepia"}),

//Relaciones bar-tapa
CREATE (caferua)-[:HASTAPA]->(anchoa),
(caferua)-[:HASTAPA]->(boqeron),
(caferua)-[:HASTAPA]->(queso),
(caferua)-[:HASTAPA]->(ajo),
(sidarta)-[:HASTAPA]->(nachos),
(sidarta)-[:HASTAPA]->(embutido),
(looking)-[:HASTAPA]->(rana),
(looking)-[:HASTAPA]->(caracoles),
(looking)-[:HASTAPA]->(empanada),
(looking)-[:HASTAPA]->(sepia),
(rinconada)-[:HASTAPA]->(cocido),
(rinconada)-[:HASTAPA]->(higado),
(rinconada)-[:HASTAPA]->(morcilla),
(rinconada)-[:HASTAPA]->(sangre),
(green)-[:HASTAPA]->(sarten),
```

Ilustración 10 - Base de datos en texto

La aplicación tiene 3 etiquetas para los nodos:

- Person, para los usuarios, con las propiedades name (nombre completo), user (el identificador en la base, es decir, el Nick o nombre de usuario) y password (guarda la contraseña). Además, tienen una propiedad fundamental para el algoritmo de recomendación que es misTapas (una lista donde se guardan todas las tapas que ha buscado un usuario, con repeticiones si ha buscado varias veces la misma, para luego poder contar qué tapa o tapas ha buscado más veces).
- Bar, para los bares, con las propiedades name (nombre del bar), address (su dirección), telephone (teléfono), web (para su página web o su cuenta en Facebook, por ejemplo). Además, tienen una serie de propiedades relacionadas con lo que ofrecen, como perros (si aceptan o no a las mascotas), futbol (si poseen los canales para ver los partidos), despedidas (si permiten celebrar estas reuniones de solteros), sidra (si tienen sidra, algo que no es tan abundante por aquí), cervezaArtesana (si tienen cerveza hecha de forma artesanal) y por último fútbolín (que indica si lo tienen).
- Tapa, para las tapas, con la propiedad tipoTapa (define lo que es).

Los bares se relacionan con sus tapas mediante una relación HASTAPA y cuando un usuario añade un bar a la lista de sus bares, se crea entre la persona y el bar una relación HASBAR. Aquí se puede ver claramente:

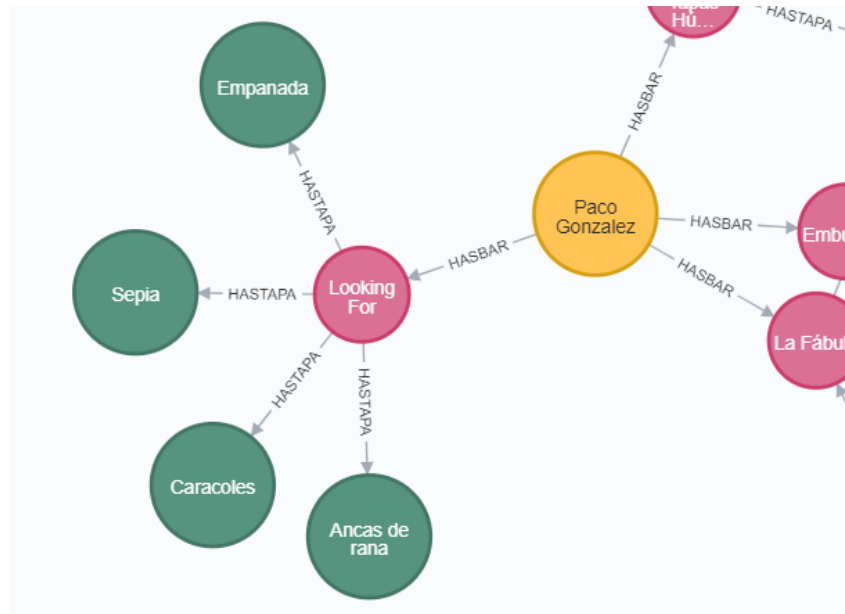


Ilustración 11 - Nodos Bar, Person y Tapa con relaciones HASTAPA y HASBAR

La base empezó siendo muy pequeña, con unos nodos usuario básicos y unos pocos bares con sus tapas y acabó teniendo 40 bares, cada uno relacionado con sus tapas, más de 70 diferentes, por lo que tiene contiene más de 110 nodos y algo más de 150 relaciones creadas. Todo esto aumenta conforme se va usando porque se pueden crear más usuarios y cada uno de ellos puede buscar bares y añadirlos a su lista con lo que se van creando más y más relaciones.

Aquí podemos ver una evolución del tamaño de la base conforme fui añadiendo nodos y relaciones:

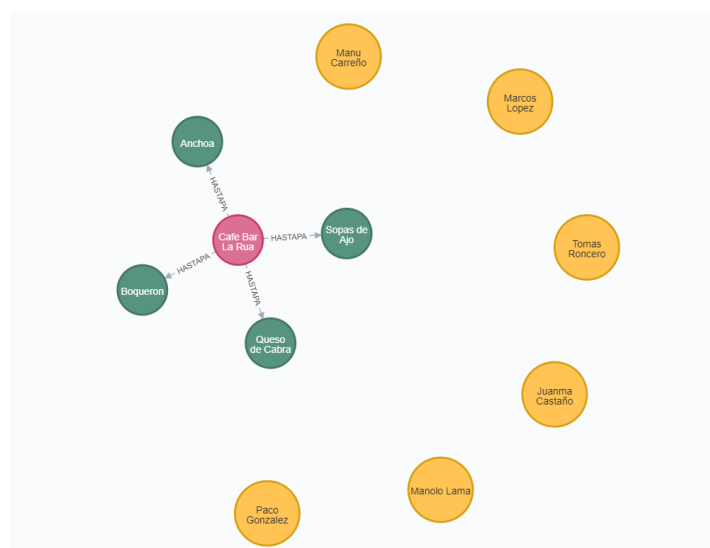


Ilustración 12 - Primer bar creado en la base

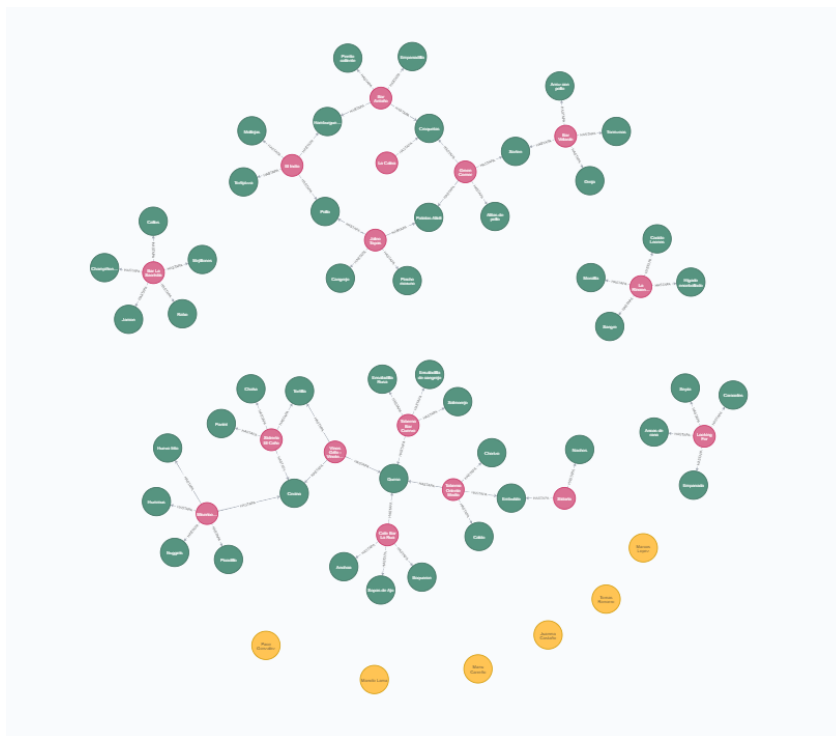


Ilustración 13 - Segunda versión de la base de datos

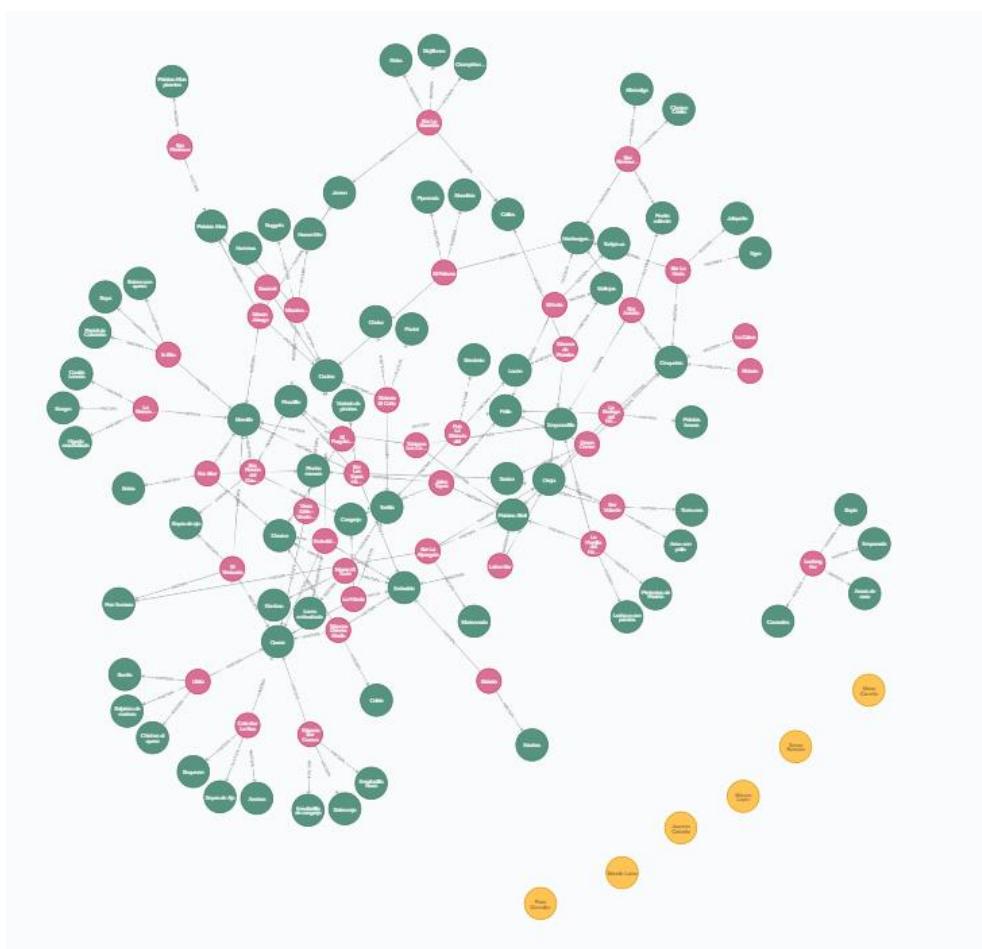


Ilustración 14 - Base de datos final

Como he dicho antes, esto es lo que está creado cuando insertas el contenido del documento de la base en Neo4j, pero conforme creas usuarios y relaciones, va evolucionando:

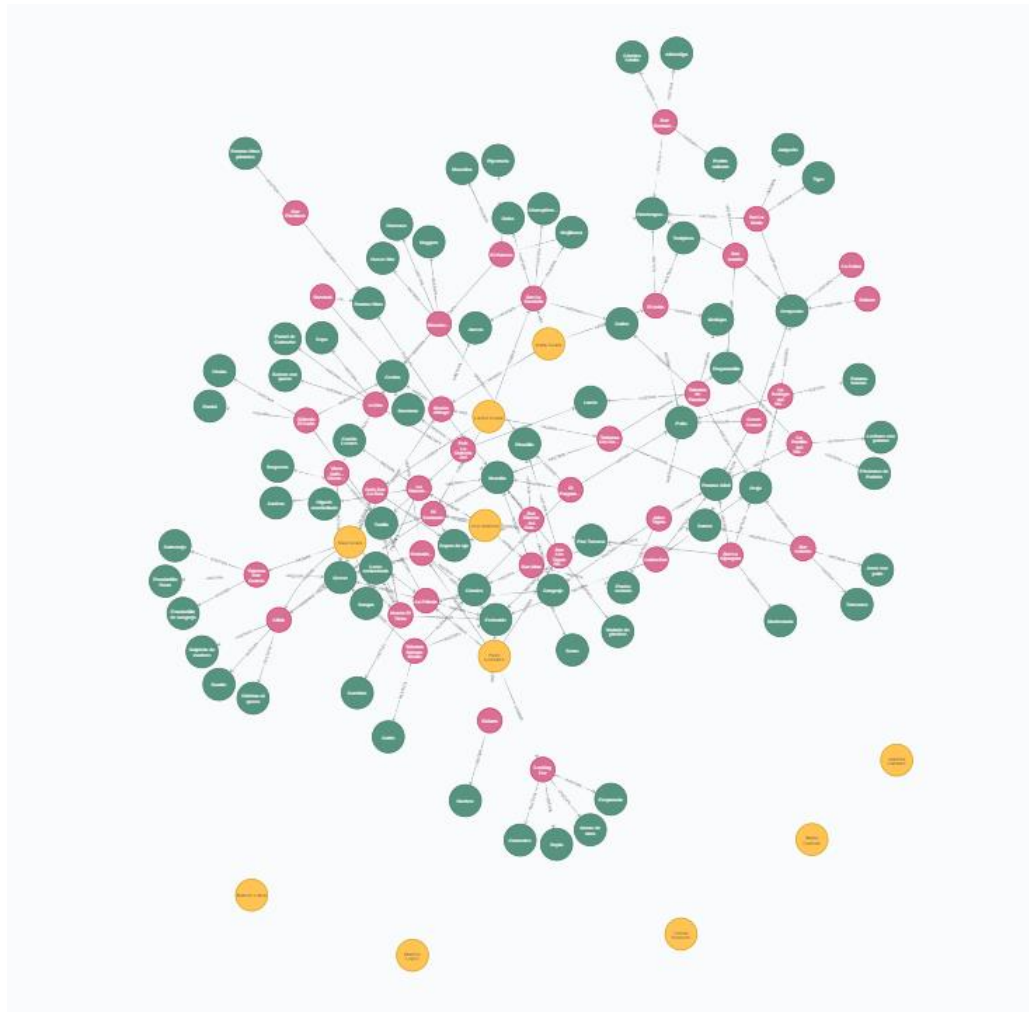


Ilustración 15 - Evolución de la base de datos conforme se utiliza la aplicación

ii. Backend y Frontend

El back y el front están muy ligados como ya he mencionado, por tanto, he decidido combinar ambos en un apartado dividido en las diferentes vistas de la aplicación y explicar qué hace cada botón y cómo responde el back a las peticiones del front.

Antes de empezar a hablar de cada una mostraré la estructura interna básica de ambos:

- Backend: Tiene una estructura sencilla, que se compone principalmente del archivo “app.js” que es donde están englobadas las consultas. En esta aplicación decidí poner todas en un solo documento, pero considerando la posibilidad de ampliar las funciones, se podrían subdividir en función del tipo de petición (comprobación del usuario/creación/búsqueda, buscar bar/añadir/borrar o buscar tapas por ejemplo, es decir, subdividir por el tipo principal de nodo buscado).

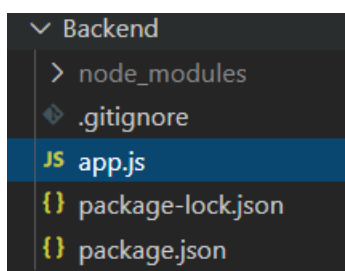


Ilustración 16 - Estructura del backend

- Frontend: La estructura aquí es mucho más complicada de explicar. En la carpeta “views” tenemos cada una de las vistas de la aplicación (las páginas por así decirlo, como cuando navegamos en una web normal). Estas vistas están indicadas en el “index.js” de la carpeta “router”, de forma que, al cargar la página principal, que es el “App.vue” nos dirige a cada una de ellas mediante el enrutamiento que hace el Vue-router. Los components son los elementos reutilizables que podemos “pegar” en cada una de ellas para evitar tener código repetido.

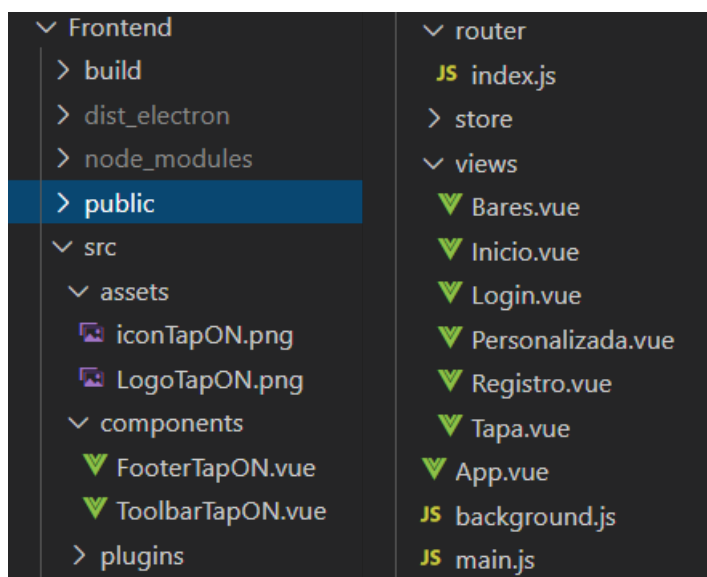


Ilustración 17 – Estructura del Frontend

Con respecto a la interfaz, tengo que mencionar previamente que no hay fotos de comida (para mostrar el luto por el cierre de los bares) y el color morado utilizado, predominante, es un intento de representar a León y a la vez, a la lucha contra la violencia machista.

Todas las funciones que contiene TapO'N son muy “gráficas”, es decir, como es un prototipo, he dejado todos los logs para poder mostrar en la consola, ya sea en el back o en el front, lo que hace la aplicación cuando pulsamos en un botón y llama a una función o hace una consulta a la base, para que quien la pruebe pueda ver con detalle su funcionalidad y saber si falla, en dónde, al igual que yo al probarla.

➤ Login/Registro

- Login:

El Login o Inicio de Sesión es la primera pantalla que ve el usuario cuando abre la aplicación. Esta vista se encuentra en el “Login.vue” dentro del frontend.

Según he pensado la app, no tiene sentido que los usuarios que no tienen una cuenta puedan acceder, porque entonces sus búsquedas quedarían guardadas solo en la parte del cliente y una vez cerrasen la aplicación no se guardarían en la base.

La parte gráfica ha sufrido algunos cambios desde mi primera idea, sin el logo y solo con la opción de entrar, hasta la versión final, que incluye un botón para registrarse y tiene los colores y el logo de la aplicación, además de un botón, el ojo, que permite mostrar u ocultar la contraseña introducida. También he añadido que si pulsamos intro sobre los campos de texto, se ejecuta la misma función que el botón de Iniciar Sesión.

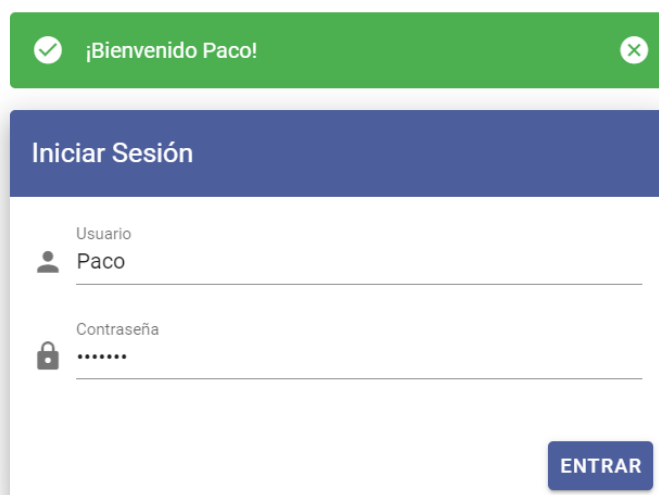


Ilustración 18 - Primera versión de la interfaz del Login



¡Bienvenido a TapO'N Paco!

Iniciar Sesión

Usuario
Paco

Contraseña
.....

REGISTRARSE INICIAR SESIÓN

Ilustración 19 - Segunda versión del Login



Iniciar Sesión

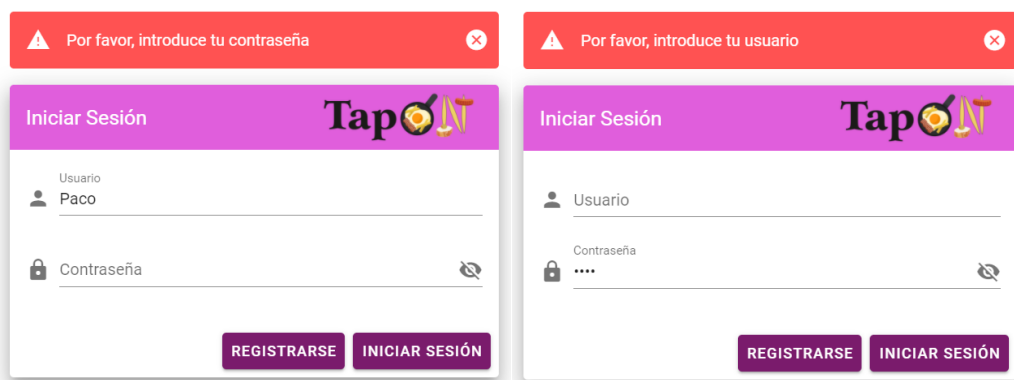
Usuario

Contraseña

REGISTRARSE INICIAR SESIÓN

Ilustración 20 - Versión final del Login

Si no introducimos alguno de los dos campos e intentamos Iniciar Sesión, nos mostrará una alerta de error e impedirá que se llame al backend.



Por favor, introduce tu contraseña

Por favor, introduce tu usuario

Iniciar Sesión

Usuario
Paco

Contraseña

REGISTRARSE INICIAR SESIÓN

Iniciar Sesión

Usuario

Contraseña
....

REGISTRARSE INICIAR SESIÓN

Ilustración 21 - Alertas de Inicio de Sesión cuando falta algún dato

Sin embargo, si introducimos ambos e iniciamos, la aplicación intentará realizar el inicio de sesión, utilizando esta función en el frontend:


```

iniciarSesion: function () {
  this.alerta = false;
  if (this.id == "" && this.contrasena == "") {
    this.tipoAlerta = "error";
    this.alerta = true;
    this.textoAlerta = "Por favor, introduce usuario y contraseña";
  } else if (this.id == "") {
    this.tipoAlerta = "error";
    this.alerta = true;
  } else if (this.contrasena == "") {
    this.tipoAlerta = "error";
    this.alerta = true;
    this.textoAlerta = "Por favor, introduce tu contraseña";
  } else {
    console.log("Intento de inicio de sesión");
    axios
      .post("http://localhost:3000/login", {
        user: this.id,
        password: this.contrasena,
      })
      .then((response) => {
        //llamada exitosa
        if (response.data.ok) {
          this.tipoAlerta = "success";
          console.log(response.data.ok + " Inicio de sesión correcto");
          this.alerta = true;
          this.textoAlerta = "¡Bienvenido a Tapo'N " + this.id + "!";
          setTimeout(() => {
            this.$router.push({
              name: "Inicio",
              params: { idUsuario: this.id },
            });
          }, 750);
        } else {
          this.tipoAlerta = "error";
          this.alerta = true;
          this.textoAlerta = "Los datos introducidos son incorrectos";
          console.log(response.data.ok + " Fallo en el inicio de sesión");
        }
      })
      .catch((alerta) => {
        //alerta
        console.log(alerta);
      });
  }
}

```

Ilustración 22 - Función que realiza el inicio de sesión desde el frontend

Esta función enviará el usuario y la contraseña introducidos a la función login del backend y si le responde con un “ok” afirmativo, significará que se ha encontrado al usuario y su contraseña era correcta, permitiéndole acceder a la aplicación. Si al buscar el usuario, alguno de los campos era erróneo, se devolverá una alerta de error por pantalla indicando que los datos eran incorrectos.

Aquí podemos ver la función del Backend, dentro del “app.js” que realiza la petición de búsqueda a la base para comprobar el usuario.

```

//Inicio de Sesión
app.post("/login", function (req, res) {
  console.log('Petición POST con params: ', req.body)
  var user = req.body.user;
  var password = req.body.password;

  var query = "MATCH (n:Person) WHERE n.user='" + user + "' AND n.password='" + password + "' RETURN n";

  const resultPromise = session.run(query);
  resultPromise.then(result => {
    //No podemos cerrar la sesión porque si no falla
    //session.close();

    if (result.records.length == 0) {
      res.send({ ok: false })
      console.log('Inicio de sesión fallido')
    } else {
      var record = result.records[0].get(0).properties.user;
      res.send({ ok: true });
      console.log('Inicio de sesión correcto con el usuario', record)
    }
  })
});

```

Ilustración 23 - Función que realiza el inicio de sesión desde el backend

Hacemos una consulta a la base usando el lenguaje Cypher que emplea Neo4j y buscamos si existe una persona con ese usuario y esa contraseña. Si no encuentra nada, la variable “result”, donde se almacena el resultado de la consulta, estará vacía y, por tanto, habrá fallado la búsqueda. Por el contrario, si devuelve un nodo de tipo Person, significará que ha conseguido encontrarlo y el usuario podrá iniciar sesión.

- Registro:

Esta pantalla es la que se muestra al usuario cuando este pulsa en “Registrarse” en la ventana del Login. Se encuentra en “Registro.vue” dentro del frontend.

Como la añadí con posterioridad, no ha sufrido muchos cambios. Su diseño se basa en el Login, añadiendo el campo de nombre completo. Aquí se podrían poner muchos tipos de datos, pero yo opté por algo básico y funcional.

Ilustración 24 - Versión final de la interfaz del Registro

Inicialmente el botón de “Registrarme” ejecutaba la función del registro y mostraba una alerta si era correcto o incorrecto y luego el usuario para iniciar sesión tenía que volver atrás con la flecha, pero finalmente he optado por permitir, en caso de que el registro sea correcto, que el usuario pueda iniciar sesión directamente.

```

} else {
  console.log("Intento de crear el usuario");
  axios
    .post("http://localhost:3000/registro", {
      user: this.id,
      password: this.contrasenya,
      name: this.nombreCompleto,
    })
    .then((response) => {
      //Llamada exitosa
      if (response.data.ok) {
        this.tipoAlerta = "success";
        console.log(response.data.ok + " creado el usuario con éxito");
        this.alerta = true;
        this.textoAlerta = "¡Usuario " + this.id + " creado!";
        setTimeout(() => {
          this.$router.push({
            name: "Inicio",
            params: { idusuario: this.id },
          });
        }, 1000);
      } else {
        this.tipoAlerta = "error";
        this.alerta = true;
        this.textoAlerta = "Fallo en el registro";
        console.log(response.data.ok + " Fallo en el registro");
      }
    })
    .catch((alerta) => {
      //alerta
      console.log(alerta);
    });
}

```

Ilustración 25 - Función que realiza el registro desde el frontend

La función para el registro que actúa en el frontend se parece mucho a la de iniciar sesión porque comprueba cada campo y solo permite que se llame al back si están todos completados (también muestra una alerta de error cuando falta alguno), por tanto, solo muestro aquí la parte más relevante.

En este caso en el back se ejecuta la función homónima. Si devuelve “ok: true” será porque se ha podido crear el usuario y será dirigido a la pantalla principal. Si no devuelve nada, será porque ha pasado lo contrario y habrá fallado el registro.

```
//Registro de usuario nuevo
app.post('/registro', function (req, res) {
  // Devolver todos si se para un json vacio
  var user = req.body.user;
  var password = req.body.password;
  var name = req.body.name;

  console.log('Petición de crear el usuario: ', req.body)

  //Ejemplo: CREATE (:Person{name:"Pepe Navarro", user:"Pepe", password:"Pepe123"})
  var query = "CREATE (n:Person{name:'" + name + "', user:'" + user + "', password:'" + password + "'})";

  console.log('Query: ', query)

  const resultPromise = session.run(query);
  resultPromise.then(result => {
    res.json({ ok: true })
    console.log('Se ha creado el usuario')
  })
});
```

Ilustración 26 - Función que realiza el registro desde el backend

Esta función se puede entender con facilidad, coge los datos que le ha pasado el front en su petición y con ellos intenta crear al usuario.

➤ Pantalla de Inicio

La pantalla de inicio es el elemento fundamental de la interfaz, la vista sobre la que todo se sostiene. Se ubica en “Inicio.vue” dentro del frontend.

Esta vista y sus funciones han sufrido muchas variaciones durante el desarrollo de la aplicación. Desde la primera versión que estaba vacía pero ya con la toolbar superior y tenía una barra inferior (que pensaba poner para promocionar la aplicación), después incluyendo los botones para hacer las búsquedas (que dirigen a las páginas que explicaré posteriormente) y, más adelante, una versión casi final con el logo y mostrando los bares del usuario.

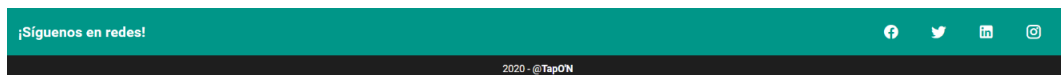
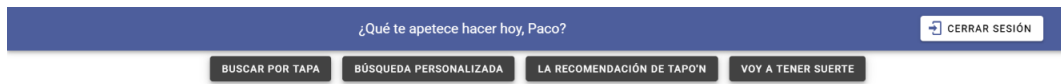


Ilustración 27 - Primera versión de la interfaz de Inicio

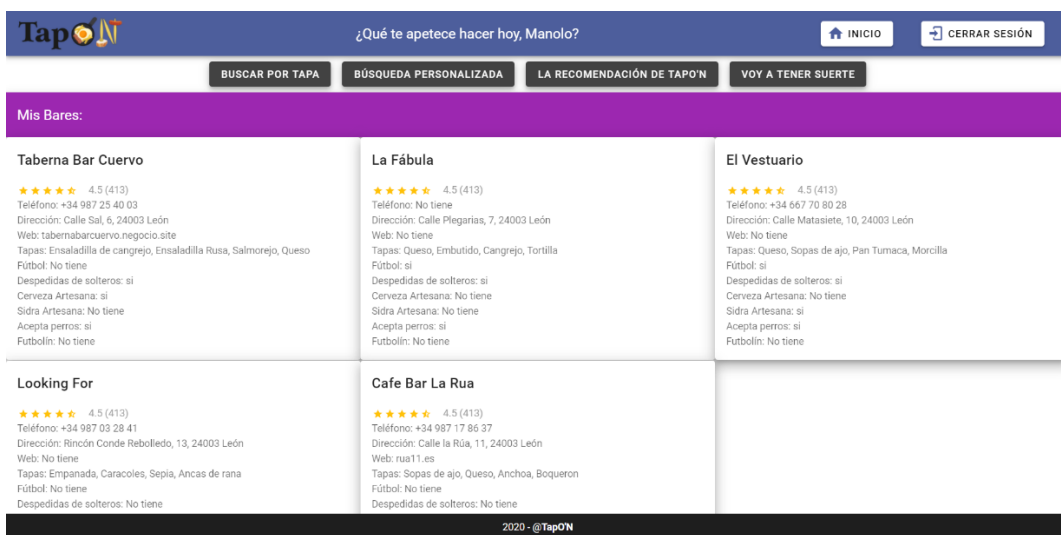


Ilustración 28 - Segunda versión de Inicio

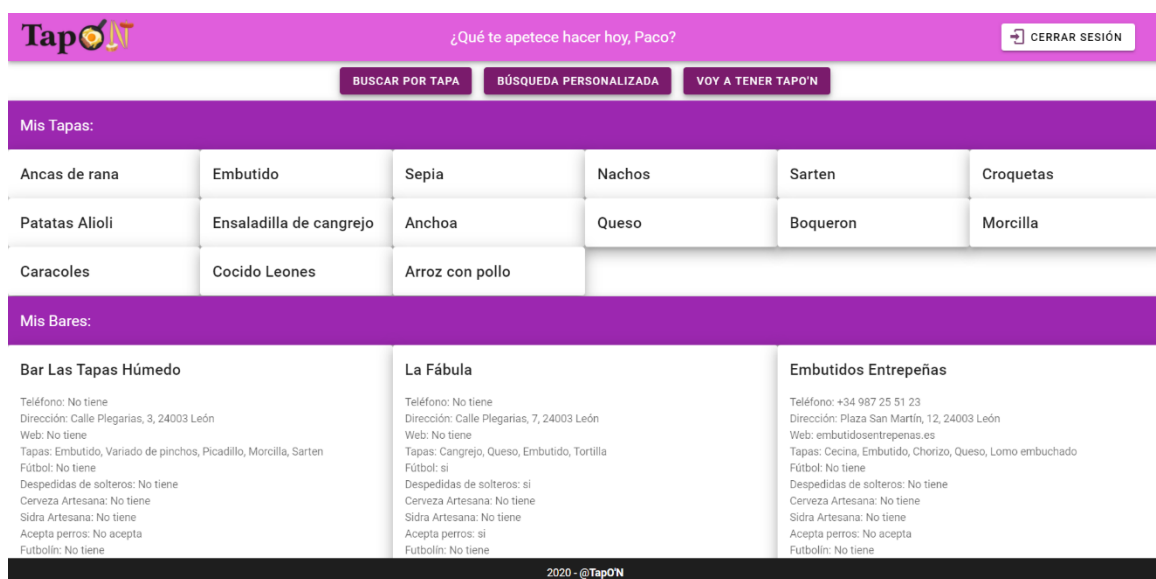


Ilustración 29 - Versión final de la pantalla de Inicio

Finalmente, la versión con los colores que escogí, los botones necesarios (solo muestra inicio en la barra superior si no estamos en la principal) que muestra bares y tapas si el usuario ha realizado búsquedas.

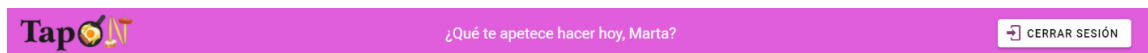


Ilustración 30 - Barra superior de la aplicación

Con respecto a sus elementos, la barra superior o toolbar no necesita explicación puesto que solo tiene la función de permitir volver a Inicio si estamos en otra pestaña o cerrar sesión, que lo que hace es devolvernos a la ventana del login ya comentada. El footer o pie no tiene ninguna función más allá de decorar.

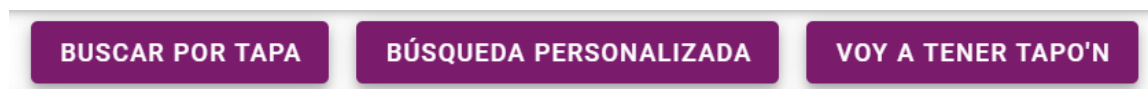


Ilustración 31 - Botones de la pantalla de Inicio

Sobre los botones de “Buscar Por Tapa”, “Búsqueda Personalizada” y “Voy a tener TapO’N”, hablaré de forma detallada en el apartado que corresponde a cada uno.

Por tanto, lo que hay que explicar en este punto es cómo funcionan “Mis Tapas” y “Mis Bares”, los cuales aparecen vacíos cuando el usuario no ha realizado búsquedas (en el caso de las tapas) o no ha añadido ningún bar a su lista (en el caso de los bares).

```
mounted: function() {
  this.cargarMisBares();
  this.cargarMisTapas();
}
```

Ambas funciones cargan a la vez que la página, es decir, cuando accedemos a Inicio, ya tenemos las dos listas en la pantalla.

Ilustración 32 - Funciones que cargan cuando llamamos a Inicio

- Mis Tapas:

Esta lista muestra las tapas que el usuario ha buscado sin contar repeticiones, es decir, muestra cada tapa distinta que el usuario ha buscado, gracias a una función que obtiene del back un array con todas las tapas que ha buscado el usuario, luego las filtra y muestra solo una aparición de cada una. Esta función es importante de cara al algoritmo de recomendación explicado en el punto 5, por tanto, volveré a hablar de ella en ese apartado.

Mis Tapas:					
Ancas de rana	Embutido	Sepia	Nachos	Sarten	Croquetas
Patatas Alioli	Ensaladilla de cangrejo	Anchoa	Queso	Boqueron	Morcilla
Caracoles	Cocido Leones	Arroz con pollo			

Ilustración 33 - Lista de Mis Tapas que aparece en Inicio

```

cargarMisTapas: function() {
  console.log("Intento de mostrar las tapas");
  axios
    .post("http://localhost:3000/misTapas", {
      user: this.idUsuario
    })
    .then(response => {
      console.log("Datos recibidos: " + response);
      //Llamada exitosa
      if (response.data.ok == true) {
        this.misTapas = response.data.datos;
        console.log(response.data + " Tapas recibidas");

        this.misTapas.forEach(el => {
          this.contador[el] = (this.contador[el] || 0) + 1;
        });
        console.log("Contador tapas: ", this.contador);
        for (var cont in this.contador) {
          this.tapasFiltro.push(cont);
          console.log("Tapa: ", cont);
        }
        console.log("El array de tapas filtrado: ", this.tapasFiltro);
      } else {
        console.log(response.data + " Fallo en la obtención de las tapas");
      }
    })
    .catch(error => {
      //Error al recoger las tapas
      console.log(error);
    });
},

```

Ilustración 34 - Función del frontend que carga las tapas de un usuario

A parte de enviar el usuario para que esa función del back pueda buscar sus tapas y obtenerlas, no solo recoge esa lista, si no que la procesa (porque contiene todas las apariciones de cada tapa) y crea otro array que contiene cada tapa concreta con su número de apariciones. Esto es muy útil porque sirve para mostrar las tapas y a la vez, esos contadores nos permiten saber cuántas veces ha buscado una concreta o cuál es la que más veces ha buscado el usuario (esto lo he utilizado para el algoritmo de recomendación), por tanto, esta función tan sencilla hace mucho más de lo que parece.

Por otro lado, tenemos la función en el back que busca las tapas del usuario. Primero formulamos la query, que consiste en buscar la propiedad del usuario en cuestión (donde se guarda cada tapa). Después, ejecutamos consulta y tenemos 3 funciones: “onNext()” que procesa cada uno de los resultados obtenidos, “onCompleted()” que será quien interactúe con la llamada del front para devolver la lista de tapas o un valor falso en caso de que no se puedan obtener, y por último “onError()” que actuará si se produce algún error durante este proceso.

```
//Buscar Mis Tapas
app.post('/misTapas', function (req, res) {
  // Devolver todos si se para un json vacio
  console.log('Petición de buscar Mis Tapas para el usuario: ', req.body.user)
  const sessionOtra = driver.session();
  var user = req.body.user;
  var tapas = [];

  var query = "MATCH (n:Person) WHERE n.user='" + user + "' RETURN DISTINCT n.misTapas AS Tapas";
  console.log("Query final: ", query)

  const resultPromise = sessionOtra.run(query).subscribe({
    onNext: function (record) {
      var tapa = record.get("Tapas");
      tapas.push(tapa);
    },
    onCompleted: function () {
      if (tapas[0] == null) {
        res.send({ ok: false })
        console.log('No se han obtenido las tapas')
      }
      else {
        console.log("Tapas: " + tapas)
        res.json({ ok: true, datos: tapas[0] });
        console.log('Tapas obtenidas correctamente')
      }
    },
    onError: function (error) {
      console.log(error + ' El usuario no tiene tapas');
    }
  });
});
```

Ilustración 35 - Función del backend que obtiene la lista de tapas de un usuario

- Mis Bares:

En este caso, es una lista que muestra los bares que ha guardado el usuario en su “lista de bares”. Esta enumeración no es más que un conjunto de relaciones que se han creado entre el usuario y los bares que ha añadido mediante la pestaña de Bares que se muestra al “Buscar por tapa” o hacer la “Búsqueda Personalizada”. Cada vez que el usuario añade un bar, se crea una relación en la base de tipo HASBAR (ver apartado Base de Datos).

Mis Bares:		
Bar Las Tapas Húmedo Teléfono: No tiene Dirección: Calle Plegarias, 3, 24003 León Web: No tiene Tapas: Embutido, Variado de pinchos, Picadillo, Morcilla, Sarten Fútbol: No tiene Despedidas de solteros: No tiene Cerveza Artesana: No tiene Sidra Artesana: No tiene Acepta perros: No acepta Fútbolín: No tiene	La Fábula Teléfono: No tiene Dirección: Calle Plegarias, 7, 24003 León Web: No tiene Tapas: Cangrejo, Queso, Embutido, Tortilla Fútbol: si Despedidas de solteros: si Cerveza Artesana: No tiene Sidra Artesana: No tiene Acepta perros: si Fútbolín: No tiene	Embutidos Entrepeñas Teléfono: +34 987 25 51 23 Dirección: Plaza San Martín, 12, 24003 León Web: embutidosentrepenas.es Tapas: Cecina, Embutido, Chorizo, Queso, Lomo embuchado Fútbol: No tiene Despedidas de solteros: No tiene Cerveza Artesana: No tiene Sidra Artesana: No tiene Acepta perros: No acepta Fútbolín: No tiene

Ilustración 36 - Lista de Mis Bares que aparece en Inicio

La función del frontend llama al back en su intento de obtener los bares para el usuario que tiene la sesión iniciada. Si la respuesta es positiva y los obtiene, carga en un array estos bares y si fuese negativa, porque no tiene ninguno guardado o bien porque ha

fallado la consulta, mostraría por consola un mensaje de error (en este caso no salta ningún aviso al usuario porque es un fallo interno y no nos interesa).

```
cargarMisBares: function() {
  console.log("Intento de mostrar los bares");
  axios
    .post("http://localhost:3000/misBares", {
      user: this.idUsuario
    })
    .then(response => {
      console.log("Datos recibidos: " + response);
      //Llamada exitosa
      if (response.data.ok == true) {
        this.bares = response.data.datos;
        console.log(response.data + " Bares recibidos");
      } else {
        this.bares = []
        console.log(response.data + " Fallo en la obtención de los bares");
      }
    })
    .catch(error => {
      //Error al recoger los bares
      console.log(error);
    });
},
```

Ilustración 37 - Función del frontend que carga los bares de un usuario

```
<v-flex md4 v-for="bar in bares" :key="bar">
  <v-card class="card-container" elevation="10">
    <template slot="progress">
      <v-progress-linear
        color="deep-purple"
        height="10"
        indeterminate
      ></v-progress-linear>
    </template>

    <v-card-title>{{ bar.name }}</v-card-title>

    <v-card-text>
      <v-row align="center" class="mx-0"> </v-row>

      <div>Teléfono: {{ bar.telephone }}</div>
      <div>Dirección: {{ bar.address }}</div>
      <div>Web: {{ bar.web }}</div>
      <div>Tapas: {{ bar.tapas.toString() }}</div>
      <div>Fútbol: {{ bar.futbol }}</div>
      <div>Despedidas de solteros: {{ bar.despedidas }}</div>
      <div>Cerveza Artesana: {{ bar.cervezaArtesana }}</div>
      <div>Sidra Artesana: {{ bar.sidra }}</div>
      <div>Acepta perros: {{ bar.perros }}</div>
      <div>Futbolín: {{ bar.futbolin }}</div>
    </v-card-text>
    <v-divider class="mx-4"></v-divider>
  </v-card>
</v-flex>
```

Ilustración 38 - Muestra de cómo se carga en la interfaz la lista de Mis Bares

Con esta lista de bares cargamos los elementos en la interfaz y vamos poniendo sus datos técnicos, las propiedades almacenadas por cada bar que hemos añadido en la base.

Aquí se ve como coincide el nombre de cada propiedad en Neo4j para los nodos de tipo Bar con el atributo que intentamos mostrar. Además, aparece un elemento más, sus tapas, que he añadido desde el backend a este array para poder mostrar todo unificado.

La función que hice en el backend ocupa mucho así que voy a desengranarla tramo a tramo para explicar bien qué hace.


```
//Buscar Mis Bares
app.post('/misBares', function (req, res) {
  // Devolver todos si se para un json vacio
  console.log('Petición de buscar Mis Bares para el usuario: ', req.body.user)
  var user = req.body.user;
  var bares = [];

  var query = "MATCH (n:Person)-[:HASBAR]->(b:Bar) MATCH (b)-[:HASTAPA]->(t:Tapa) WHERE n.user='" + user + "' RETURN DISTINCT b as Bar, collect(t.tipoTapa) as Tapas";
  console.log("Query final: ", query)
```

Ilustración 39 - Primera parte de la función que carga los bares en el backend

En esta primera parte, lo que hacemos es preparar la consulta, con el usuario obtenido del frontend, para buscar todos los bares con los que tiene relación y, además, obtenemos las tapas que tiene cada bar agrupadas mediante la función “collect()”.

```
const resultPromise = session.run(query).subscribe({
  onNext: function (record) {
    var bar = record.get("Bar").properties;
    if (bar.name !== undefined) {
      if (bar.address === undefined) {
        bar.address = "No tiene"
      }
      if (bar.telephone === undefined) {
        bar.telephone = "No tiene"
      }
      if (bar.web === undefined) {
        bar.web = "No tiene"
      }
      if (bar.perros === undefined) {
        bar.perros = "No acepta"
      }
      if (bar.futbolin === undefined) {
        bar.futbolin = "No tiene"
      }
      if (bar.sidra === undefined) {
        bar.sidra = "No tiene"
      }
      if (bar.cervezaArtesana === undefined) {
        bar.cervezaArtesana = "No tiene"
      }
      if (bar.despedidas === undefined) {
        bar.despedidas = "No tiene"
      }
      if (bar.futbol === undefined) {
        bar.futbol = "No tiene"
      }
      bar.tapas = record.get("Tapas")
      bares.push(bar);
    }
  },
```

Ilustración 40 - Tramo de la función de cargar los bares en el backend que procesa cada bar encontrado

Ahora viene la clave de esta función, la ejecución de la consulta y dentro de ella, 3 subfunciones, por así decirlo. En primer lugar, “onNext()” procesa cada uno de los resultados obtenidos en la búsqueda, es decir, en este caso, cada nodo Bar. Como hay campos que pueden ser nulos (undefined en caso de Neo4j) tenemos que analizar cada uno y en caso de que lo sea, cambiar este valor por uno más “entendible” para el usuario y que además permita mostrarlos desde la interfaz sin que provoque fallos (que los provocaría en caso de intentar acceder a una de sus propiedades y que fuese undefined). Además de eso, como también hemos recogido las tapas que están relacionadas con cada bar, nos inventamos un nuevo campo y metemos esta lista de tapas en cada bar y después añadimos cada uno a la lista general de bares.

```
onCompleted: function () {
  if (bares.length == 0) {
    res.send({ ok: false })
    console.log('No se han obtenido los bares')
  }
  else {
    console.log('Se han encontrado ' + bares.length + ' bares')
    for (var i = 0; i < bares.length; i++) {
      console.log("Bar: " + bares[i].name)
      console.log("Bar: " + bares[i].futbol)
      console.log("Bar: " + bares[i].perros)
      console.log("Bar: " + bares[i].despedidas)

      for (var j = 0; j < bares[i].tapas.length; j++) {
        bares[i].tapas[j] = " " + bares[i].tapas[j]
      }
      console.log("Bar: " + bares[i].tapas)
    }
    res.json({ ok: true, datos: bares });
    console.log('Bares obtenidos correctamente')
  }
},
onError: function (error) {
  console.log(error + ' El usuario no tiene bares');
}
});
});
```

Ilustración 41 - Último tramo de la función que actúa en el backend para cargar los bares

Después tenemos “onCompleted()”, que actúa cuando la consulta ha finalizado y ya tenemos todos los resultados. Aquí es cuando devolveremos la lista de los bares si todo ha ido bien o devolveremos un “ok: false” si la lista estuviera vacía. Los logs que se ven son una muestra para ver a través de la terminal si se ha cambiado correctamente el valor de los campos correspondientes a las propiedades que pueden ser nulas. Por último “onError()” actuaría en caso de que se produjese algún fallo durante el proceso.

Además de mostrar los bares, desde la interfaz de inicio podemos borrarlos de la lista obtenida mediante el botón “Borrar Bar de Mis Bares”.

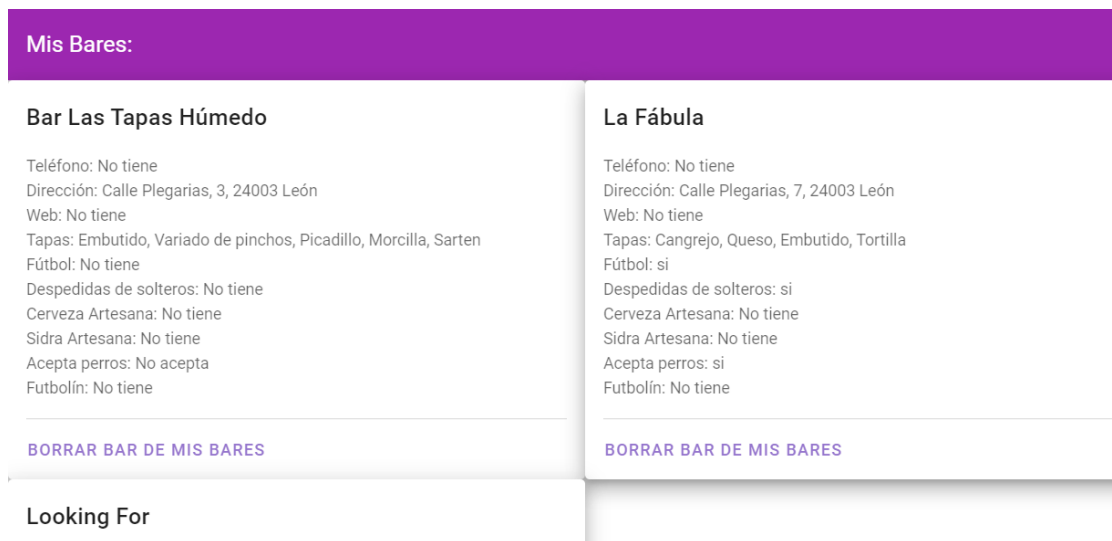


Ilustración 42 - Botón de Borrar Bar que aparece en la lista de los bares en Inicio

Esta es otra función del front que, al pulsar ese botón de borrar, llama al back con el usuario y el nombre del bar seleccionado. Muestra diferentes alertas al usuario en función de si consigue o no su propósito y, además, actualiza la lista en caso de borrar el bar.

```

borrarBar: function(nombreBar) {
  //Primero comprobamos que se ha seleccionado un bar
  if (nombreBar != "") {
    this.tipoAlerta = "success";
    this.alerta = true;
    this.textoAlerta2 = "Se ha seleccionado el bar " + nombreBar;

    console.log("Intento de borrar el bar");
    axios
      .post("http://localhost:3000/borrarBar", {
        bar: nombreBar,
        user: this.idUsuario
      })
      .then(response => {
        console.log("Datos recibidos: " + response.data.ok);
        //Llamada exitosa
        if (response.data.ok == true) {
          this.tipoAlerta = "success";
          this.alerta = true;
          this.textoAlerta2 = "¡Bar borrado correctamente!";

          console.log("Se ha borrado el bar");
          //Si se borra correctamente, cargamos de nuevo todos los bares
          this.cargarMisBares();
        } else {
          this.tipoAlerta = "error";
          this.alerta = true;
          this.textoAlerta2 = "No se ha podido borrar el bar";

          console.log("Fallo en el borrado del bar");
        }
      })
  }
}

```

Ilustración 43- Función de borrar bar que actúa en el frontend

Con respecto al backend, en este caso la función es más sencilla. “Simplemente” busca la relación entre el usuario y el bar que hemos pasado y la borra si existe, algo que no debe dar problemas puesto que el usuario solo ve sus bares guardados y, por tanto, tiene que existir esta relación.

```
//Método que borra un bar de la lista de mis bares
app.post('/borrarBar', function (req, res) {
  // Devolver todos si se para un json vacío
  var bar = req.body.bar;
  var user = req.body.user;
  console.log('Petición de borrar el bar: ', req.body)

  //Usamos Merge, si ya existía la relación no la crea, si no existía, sí
  var query = "MATCH (n:Person)-[rel:HASBAR]->(b:Bar) WHERE n.user='" + user + "' AND b.name='" + bar + "' DELETE rel";

  console.log('Query: ', query)

  const resultPromise = session.run(query);
  resultPromise.then(result => {
    res.json({ ok: true })
    console.log('Se ha borrado la relación')
  })
});
```

Ilustración 44 - Función del backend que borra el bar seleccionado

➤ Buscar Por Tapa

Cuando el usuario pulsa “Buscar por Tapa” en la página principal, se muestra esta vista, que se encuentra en el frontend con el nombre de “Tapa.vue”.

Al igual que la pantalla de Inicio, ha sufrido muchos cambios durante el desarrollo. Al principio era solo una lista que mostraba todas las tapas, pero era muy difícil encontrar lo que queríamos al no poder filtrar.

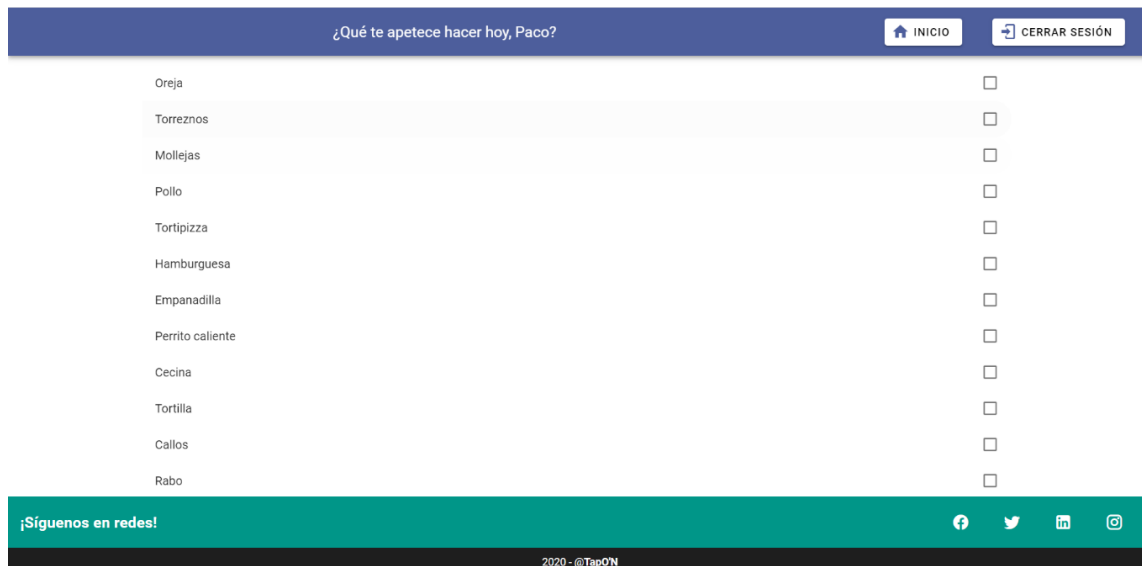


Ilustración 45 - Primera versión de la interfaz de Buscar Tapa

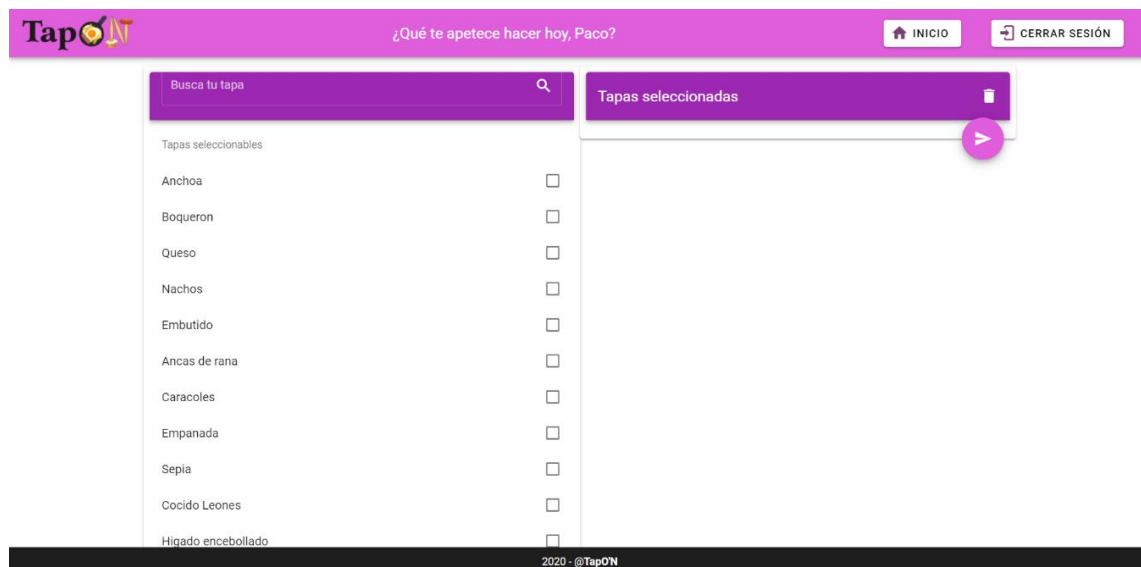


Ilustración 46 - Versión final de la interfaz de Buscar Tapa

En la versión final, además de la lista he puesto un buscador que permite filtrar por texto introducido y de esta forma hacer más sencilla la búsqueda. También he puesto una lista a la derecha que permite ver las seleccionadas y borrar todas si no nos convence lo que hemos seleccionado. Se puede buscar pulsando la lupa o con la tecla intro. Para volver a tener todas, si dejamos vacío el campo y pulsamos, podemos volver a obtener la lista completa.

Aquí se puede ver cómo funciona el filtrado y cómo se van añadiendo a la lista de tapas seleccionadas las que hemos marcado:

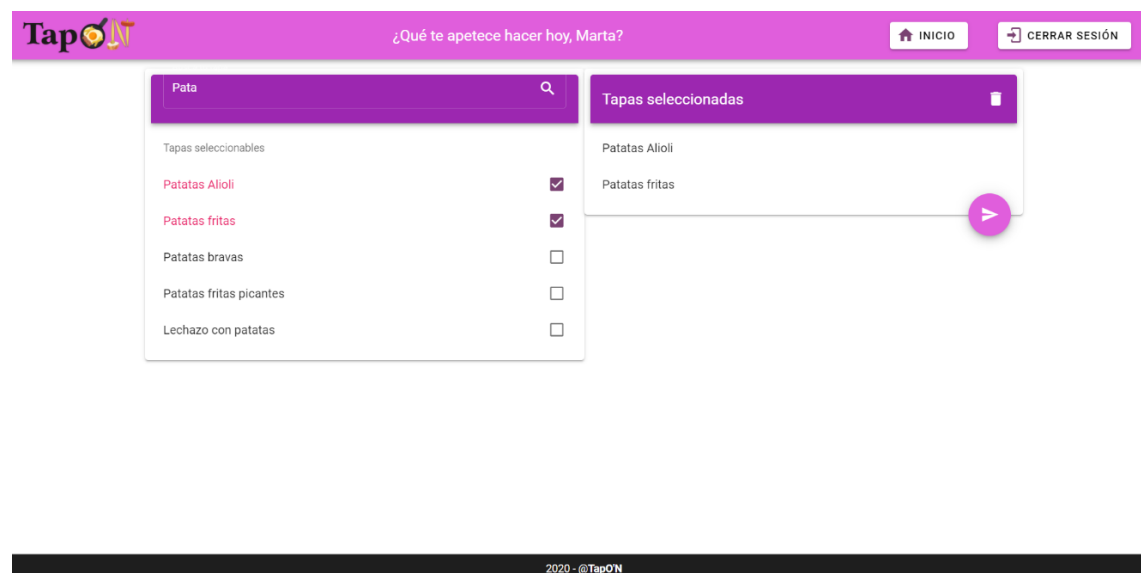


Ilustración 47 - Muestra de cómo funciona el filtrado y la selección de tapas

La lista general de tapas aparece cargada en la pantalla como ocurre con Mis Tapas y Mis Bares en la pantalla principal. Podemos cargar esta lista con esta función en el front, que además es doble, puesto que actúa en función de lo que haya en el campo de búsqueda como ya he mencionado. Si está vacío, como al cargar la página, muestra todas:

```

buscarTapas: function() {
  //Si está vacío, obtenemos todas las tapas
  if (this.buscar == "") {
    //Obtenemos todas las tapas por pantalla al cargar la página
    console.log("Intento de mostrar las tapas");
    axios
      .post("http://localhost:3000/tapas", {})
      .then(response => {
        console.log("Datos recibidos: " + response);
        //Llamada exitosa
        if (response.data.ok == true) {
          this.tapas = response.data.datos;
          console.log(response.data.datos + " Tapas recibidas");
        } else {
          console.log(
            response.data.datos + " Fallo en la obtención de las tapas"
          );
        }
      })
      .catch(error => {
        //Error al recoger las tapas
        console.log(error);
      });
  }
}

```

Ilustración 48 - Función que carga las tapas en el frontend

Para este tramo, el front llama a la función “tapas” del back que devuelve el nombre de todas las tapas que están en la base:

```

//Obtener todas las tapas
app.post('/tapas', function (req, res) {
  // Devolver todas si se para un json vacío
  var query = "MATCH (n:Tapa) RETURN n.tipoTapa";

  const resultPromise = session.run(query);
  resultPromise.then(result => {
    if (result.records.length == 0) {
      res.send({ ok: false })
      console.log('No se han obtenido las tapas')
    }
    else {
      var tapas = result.records;
      var tapasFiltro = [];

      //Sacar el tipo de tapa -> devuelve Label: [values]
      for (var i = 0; i < tapas.length; i++) {
        console.log('Tapa: ', tapas[i]._fields[0])
        tapasFiltro.push(tapas[i]._fields[0])
      }
      res.json({ ok: true, datos: tapasFiltro });
    }
  });
});

```

Ilustración 49 - Función del backend que obtiene todas las tapas de la base

Como en anteriores funciones, si sale todo bien, devuelve la lista con todas las tapas que están en nuestra base y, en caso contrario, devolvería un falso que interpretamos como error. Es una consulta sencilla que devuelve todos los nodos de tipo Tapa.

En caso de que introdujésemos texto en el campo de búsqueda, se ejecutaría este otro trozo de la función de búsqueda en el front:

```
//Solo buscamos si la longitud no está vacía
else if (this.buscar != "") {
  console.log("Ha pulsado enter con valor de búsqueda: ", this.buscar);
  console.log("Intento de buscar tapa");
  axios
    .post("http://localhost:3000/buscarTapas", {
      tapa: this.buscar
    })
    .then(response => {
      console.log("Datos recibidos: " + response.data.ok);
      //Llamada exitosa
      if (response.data.ok == true) {
        this.alerta1 = false;
        this.tapas = response.data.datos;
        console.log("Se han encontrado las tapas");
      } else {
        this.tipoAlerta = "error";
        this.alerta1 = true;
        this.textoAlerta = "No hay tapas con esos criterios";
        console.log("Fallo en la búsqueda de las tapas");
      }
    })
    .catch(error => {
      //Error al recoger los bares
      console.log(error);
    });
}
```

Ilustración 50 - Parte de la función del frontend que busca tapas filtrando por texto

Es decir, el front mandaría el valor introducido por el usuario al back para que busque si hay tapas que contengan ese texto.

```
//Búsqueda de tapas por texto
app.post('/buscarTapas', function (req, res) {
  // Devolver todos si se para un json vacío
  console.log('Petición de buscar tapas que contengan: ', req.body.tapa)
  var tapa = req.body.tapa;

  var query = "MATCH (t:Tapa) WHERE toLower(t.tipoTapa) CONTAINS toLower(' ' + tapa + ' ') RETURN t.tipoTapa";

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

    if (result.records.length == 0) {
      res.send({ ok: false })
      console.log('No se han obtenido las tapas')
    }
    else {
      var tapas = result.records;
      var tapasFiltro = [];

      //Sacar el tipo de tapa -> devuelve label: [values]
      for (var i = 0; i < tapas.length; i++) {
        console.log('Tapa: ', tapas[i]._fields[0])
        tapasFiltro.push(tapas[i]._fields[0])
      }
      res.json({ ok: true, datos: tapasFiltro });
    }
  })
});
```

Ilustración 51 - Función del backend que obtiene la lista de tapas filtradas por texto

En el back, recibimos ese texto y buscamos tapas que lo contengan. Pasamos tanto el texto del usuario como el nombre de cada tapa de la base a minúsculas con la función “toLowerCase()” de Neo4j, de forma que así nos aseguramos de que no hay problemas al comparar y aunque el usuario escribiese todo en mayúsculas o minúsculas no importaría.

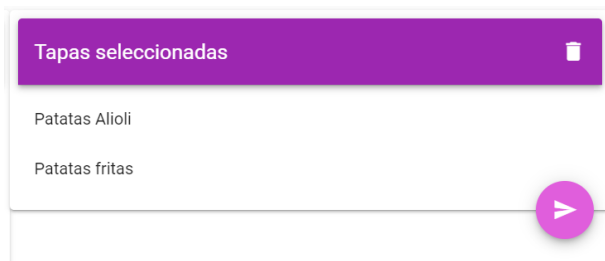


Ilustración 52 - Ejemplo de tapas seleccionadas

Una vez seleccionadas las tapas, el botón que se encuentra debajo de la lista de “Tapas Seleccionadas” nos permite realizar la búsqueda de los bares, la cual se ejecuta realmente en esta ventana y no en la de Bares (que se encarga de recibir la lista encontrada).

La función que actúa en el front enviará esa lista de tapas seleccionadas al back. Primero comprobará que se ha seleccionado al menos una tapa y mostrará una alerta de error en caso de que el usuario no haya seleccionado ninguna. Si se cumple y ha seleccionado al menos una, llamará al back y, en caso de encontrar bares para esas tapas, lo indicará con una alerta de éxito, en verde, y empujará la vista de Bares con la lista que ha encontrado. Si no encuentra ninguno, mostrará una alerta de error. Además de todo esto, incluso aunque la búsqueda no encontrase ningún bar, las tapas seleccionadas las guardamos y las añadimos a las acumuladas del usuario con otra función.

```
//agregamos las tapas buscadas
this.agregarMisTapas();
console.log("Intento de buscar bares");
axios
  .post("http://localhost:3000/buscarBares", {
    tapas: this.tapasSeleccionadas,
    user: this.idUsuario
  })
  .then(response => {
    console.log("Datos recibidos: " + response.data.ok);
    //Llamada exitosa
    if (response.data.ok == true) {
      this.bares = response.data.datos;
      this.tipoAlerta = "success";
      this.alerta2 = true;
      this.textoAlerta = "¡Bares encontrados!";

      console.log("Se han encontrado los bares");
      setTimeout(() => {
        this.$router.push({
          name: "Bares",
          params: { idUsuario: this.idUsuario, bares: this.bares }
        });
      }, 750);
    } else {
      this.tipoAlerta = "error";
      this.alerta2 = true;
      this.textoAlerta =
        "No se han encontrado bares con las tapas seleccionadas";
      console.log("Fallo en la búsqueda de los bares");
    }
  })
  .catch(() => {
    console.log("Error al buscar bares");
  });
```

Ilustración 53 - Función del frontend que ejecuta la búsqueda de bares con las tapas seleccionadas

La función de la búsqueda de los bares en el back se parece a la que actúa en Inicio para mostrar los bares (también realiza un procesamiento de las búsquedas), por tanto, mostraré solo la parte inicial que es la más relevante.

En primer lugar, tengo que decir que como no hay tantos bares en la base, las búsquedas no son “estrictas”. ¿Qué quiere decir esto? Si buscamos bares habiendo seleccionado 4 tapas, por ejemplo, la aplicación devuelve bares donde haya al menos una de esas cuatro tapas, porque probablemente no haya ninguno guardado que tenga justo todas esas tapas. Para poder hacer ese tipo de búsquedas habría que tener una base muy completa, algo que no podemos plantear en poco tiempo siendo esto un prototipo construido a mano. No obstante, he dejado comentada la construcción de cómo habría que formular la consulta de hacerlo de ese modo.

```
//Buscar los bares en funcion de las tapas seleccionadas
app.post('/buscarBares', function (req, res) {
  console.log('Petición de buscar bares con las tapas: ', req.body)
  const sessionOtra = driver.session();
  var tapas = req.body.tapas;
  var user = req.body.user;
  var bares = [];
  //Primera prueba: devolvemos todos los bares que tienen alguna de las tapas seleccionadas porque tenemos pocos en la base
  //MATCH (b:Bar)-[:HASTAPA]->(t:Tapa) MATCH (b)-[:HASTAPA]->(t2:Tapa) WHERE t.tipoTapa IN ['Picadillo', 'Morcilla', 'Cangrejo'] RETURN b as Bar

  //Para coger solo los que tienen las tapas seleccionadas, ejemplo con 3 tapas:
  //MATCH (b:Bar)-[:HASTAPA]->(t1:Tapa)
  //MATCH (b)-[:HASTAPA]->(t2:Tapa)
  //MATCH (b)-[:HASTAPA]->(t3:Tapa)
  //MATCH (b)-[:HASTAPA]->(t:Tapa)
  //WHERE t1.tipoTapa = 'Picadillo' AND t2.tipoTapa='Morcilla' AND t3.tipoTapa ='Cangrejo' RETURN b,t1,t2,t3,t

  var query = "MATCH (n:Person), (b:Bar)-[:HASTAPA]->(t:Tapa) MATCH (b)-[:HASTAPA]->(t2:Tapa) WHERE t.tipoTapa IN [";
  console.log("Query parcial: ", query)
  for (var i = 0; i < tapas.length; i++) {
    if (i + 1 < tapas.length) {
      query = query.concat("'" + tapas[i] + "',");
    }
    else if (i + 1 == tapas.length) {
      query = query.concat("'" + tapas[i] + "'");
    }
  }
  console.log("Query prefinal: ", query)
  query = query.concat("] AND n.user='" + user + "' AND NOT exists((n)-[:HASBAR]->(b)) RETURN DISTINCT b as Bar, collect(t2.tipoTapa) as Tapas");
  console.log("Query final: ", query)
});
```

Ilustración 54 - Función del backend que busca los bares con las tapas seleccionadas

En esta función la clave es la construcción de la query, donde vamos añadiendo tantas tapas como haya seleccionado el usuario en la pantalla y vamos concatenando, por eso la parte más importante es el “WHERE tipoTapa IN [” que es donde va a buscar las tapas. La otra parte fundamental es el añadido de buscar bares que no haya añadido el usuario a su lista, es decir, donde no exista la relación “(usuario)-[HASBAR]-(bar)” porque no sería interesante que nos muestren de nuevo un bar que ya tengamos guardado al buscar nuevos.

Como he dicho, la parte inferior cuando realiza la consulta se parece mucho a la de Inicio para los bares, porque procesa cada resultado y cambia campos no definidos. Si encuentra los bares, se los devolverá a la interfaz de las tapas, que llamará a la siguiente vista que expodré, Bares, con ese array.

➤ Bares

Esta pestaña es un complemento de ayuda que muestra los bares que hemos buscado, ya sea en “Buscar por Tapa”, “Búsqueda Personalizada” o, como explicaré posteriormente, en la recomendación que realiza la aplicación personalizada para cada usuario. Se encuentra en el frontend en “Bares.vue”.

Esta vista también ha sufrido cambios, aunque no muy notables. Tenía pensada la idea de cómo mostrarlos y ya la primera se parece a la versión final. Consta de una lista con los bares encontrados mostrados en forma de tarjetas con sus propiedades, al igual que en Inicio, pero cambiando el botón de “Borrar” por el de “Añadir”.

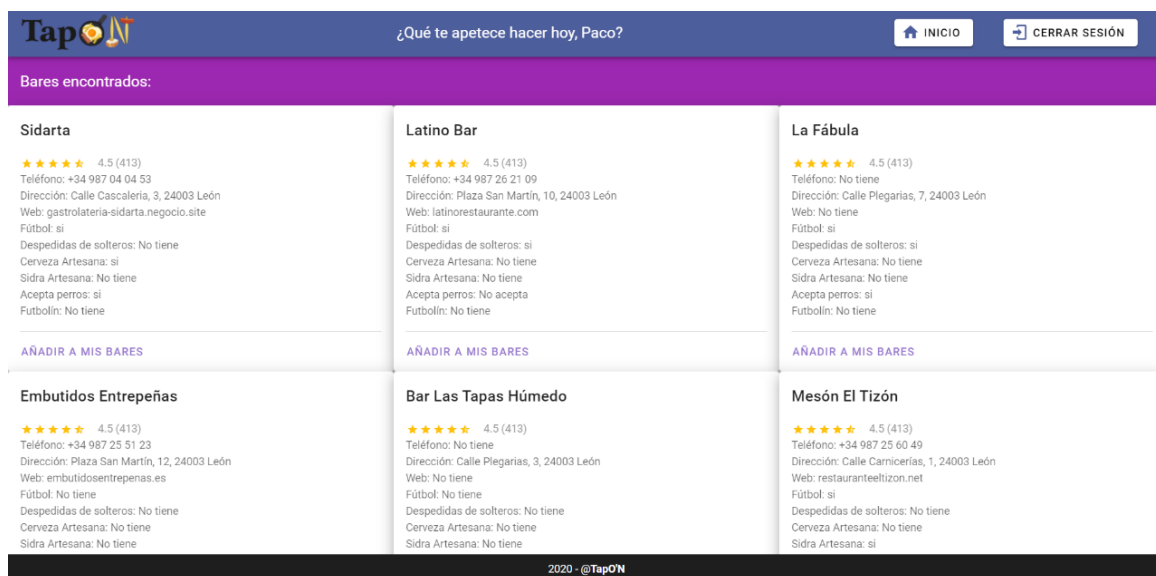


Ilustración 55 - Primera versión de la vista de Bares

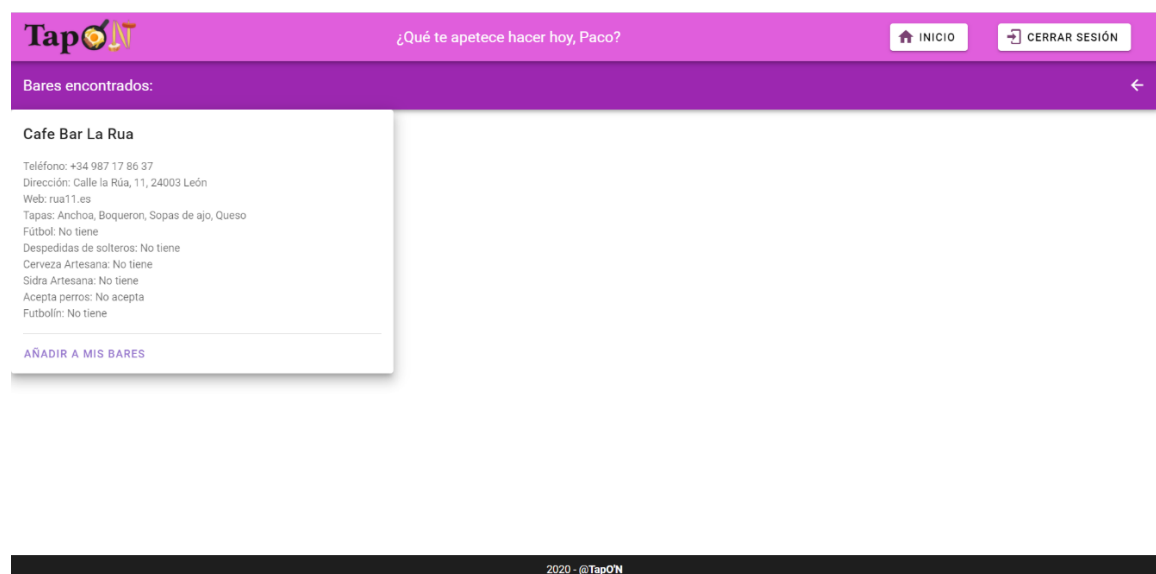


Ilustración 56 - Versión final de la vista de los Bares

La función principal de esta vista es mostrar los bares y poder agregarlos a los previamente guardados. Como recibe la lista de las vistas mencionadas, que son quienes realizan la búsqueda, solo cuenta con dos funciones principales, una que devuelve al usuario a la vista anterior (con la flecha de volver arriba a la derecha) y lleva una serie de valores para saber de cuál de todas las búsquedas se trataba; y otra que permite agregar los bares al pulsar en el botón, de uno en uno siempre. La lista no se vacía aunque añadamos un bar.

```
agregarBar: function(nombreBar) {
  //Primero comprobamos que se ha seleccionado un bar
  if (nombreBar !== "") {
    this.tipoAlerta = "success";
    this.alerta = true;
    this.textoAlerta = "Se ha seleccionado el bar " + nombreBar;

    console.log("Intento de añadir el bar");
    axios
      .post("http://localhost:3000/agregarBar", {
        bar: nombreBar,
        user: this.idUsuario
      })
      .then(response => {
        console.log("Datos recibidos: " + response.data.ok);
        //Llamada exitosa
        if (response.data.ok === true) {
          this.tipoAlerta = "success";
          this.alerta = true;
          this.textoAlerta = "¡Bar añadido correctamente!";

          console.log("Se ha añadido el bar");
        } else {
          this.tipoAlerta = "error";
          this.alerta = true;
          this.textoAlerta = "No se ha podido añadir el bar";

          console.log("Fallo en la agregación del bar");
        }
      })
      .catch(error => {
        //Error al añadir el bar
      })
  }
}
```

Ilustración 57 - Función que permite agregar bares a la lista de Mis Bares en el frontend

Esta función es la inversa del borrado que podemos hacer en Inicio. También envía al backend el usuario y el nombre del bar seleccionado y responde al usuario en función del éxito o el fracaso en esta misión mediante alertas.

```
//Método que agrega un bar a la lista de mis bares
app.post('/agregarBar', function (req, res) {
  // Devolver todos si se para un json vacío
  var bar = req.body.bar;
  var user = req.body.user;
  console.log('Petición de añadir el bar: ', req.body)

  //Usamos Merge, si ya existía la relación no la crea, si no existía, sí
  var query = "MATCH (n:Person), (b:Bar) WHERE n.user='" + user + "' AND b.name='" + bar + "' MERGE (n)-[:HASBAR]->(b)";

  console.log('Query: ', query)

  const resultPromise = session.run(query);
  resultPromise.then(result => {
    res.json({ ok: true })
    console.log('Se ha creado la relación')
  })
});
```

Ilustración 58 - Función del backend para agregar un bar a la lista de Mis Bares

En el back, la función también es muy parecida a la del borrado, pero creando la relación entre el usuario y el bar (de tipo “HASBAR”) en lugar de borrarla.

➤ Búsqueda Personalizada

Cuando el usuario pulsa “Búsqueda Personalizada” en la pantalla principal, la aplicación le muestra esta vista, que es una ampliación de Buscar Tapa. Se encuentra en “Personalizada.vue” en el frontend.

Esta vista pertenece a la última parte del desarrollo por lo que gráficamente no experimentó muchos cambios. Muestra por pantalla dos listas cargadas previamente, la lista de las tapas (de la misma forma y con el mismo funcionamiento que el explicado anteriormente) y una lista de complementos que puede seleccionar el usuario, como que tenga cerveza artesana o cuente con canales de fútbol. Estos complementos corresponden con las propiedades de los bares comentadas anteriormente.

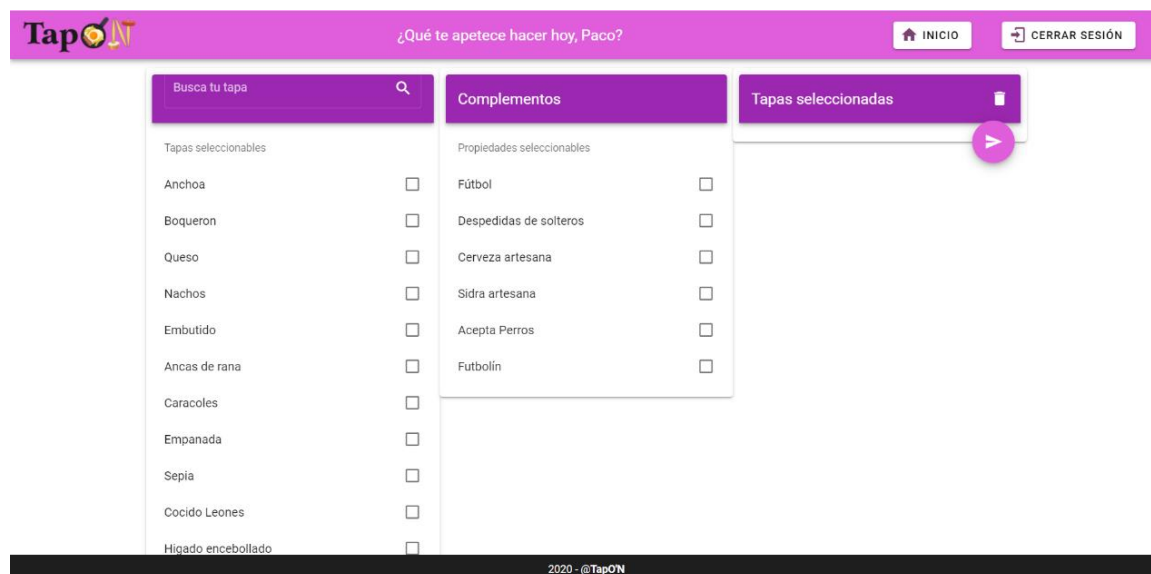


Ilustración 59 - Versión final de la interfaz de Búsqueda Personalizada

Las funciones que ofrece esta vista son las mismas que la búsqueda de las tapas mostrada con anterioridad, pero ampliadas con la lista nueva, de forma que las funciones del front y del backend se amplían para poder abarcar los nuevos parámetros.

La diferencia que hay ahora es que todos los complementos que añadamos provocarán que la búsqueda sí sea “estricta” en ese sentido, es decir, no va a buscar únicamente los bares que tengan todas las tapas que seleccionemos, pero en el caso de las propiedades adicionales, sí. Por lo tanto, si marcamos “Fútbol” y “Despedidas” solo nos devolverá bares que tengan ambas. Estas propiedades son menos y mucho más generales que las tapas, por lo que he pensado que es mejor hacer la búsqueda de esta manera.

En el caso del front, en la función de búsqueda se añaden las propiedades seleccionadas en la llamada y el usuario debe seleccionar al menos una en cada una de las dos listas, es decir, por lo menos una tapa y un complemento, de lo contrario le aparecerán alertas para que lo haga.

```
//agregamos las tapas buscadas
this.agregarMisTapas();
console.log("Intento de buscar bares");
axios
  .post("http://localhost:3000/buscarBaresPersonalizados", {
    tapas: this.tapasSeleccionadas,
    propiedades: this.propiedadesSeleccionadas,
    user: this.idUsuario
  })
  .then(response => {
    console.log("Datos recibidos: " + response.data.ok);
    //Llamada exitosa
    if (response.data.ok == true) {
      this.bares = response.data.datos;
      this.tipoAlerta = "success";
      this.alerta2 = true;
      this.textoAlerta = "¡Bares encontrados!";

      console.log("Se han encontrado los bares");
      setTimeout(() => {
        this.$router.push({
          name: "Bares",
          params: {
            idUsuario: this.idUsuario,
            bares: this.bares,
            personalizada: "SI"
          }
        });
      }, 750);
    }
  });
```

Ilustración 60 - Función del frontend para buscar bares en función de las tapas y los complementos seleccionados

Por otro lado, en el backend, la función es como la de buscar bares solo con tapas, pero añadiendo las propiedades en caso de que las hayamos seleccionado, lo cual comprobamos con un bucle que recorre la lista de los complementos y añade a la query el parámetro en caso de que esté uno de ellos. Como se parece mucho a la función ya mostrada, solo pondré el trozo que filtra y permite identificar a estas búsquedas más complejas.

```
//Ahora, en función de la propiedad añadida, hacemos AND
query = query.concat(" ");
for (var j = 0; j < propiedades.length; j++) {
    if (propiedades[j] == 'Fútbol') {
        query = query.concat("AND b.futbol='si' ");
    }
    if (propiedades[j] == 'Acepta Perros') {
        query = query.concat("AND b.perros='si' ");
    }
    if (propiedades[j] == 'Despedidas de solteros') {
        query = query.concat("AND b.despedidas='si' ");
    }
    if (propiedades[j] == 'Cerveza artesana') {
        query = query.concat("AND b.cervezaArtesana='si' ");
    }
    if (propiedades[j] == 'Sidra artesana') {
        query = query.concat("AND b.sidra='si' ");
    }
    if (propiedades[j] == 'Futbolín') {
        query = query.concat("AND b.futbolin='si' ");
    }
}
```

Ilustración 61 - Filtro que utiliza el backend para buscar bares con tapas y complementos

5. Algoritmo empleado – Voy a tener TapO'N

En este punto explicaré el algoritmo de recomendación que actúa cuando el usuario pulsa el botón de “Voy a tener TapO'N”.

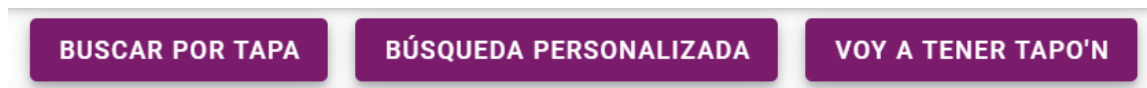


Ilustración 62 - Botones de la pantalla de Inicio

Este algoritmo se puede resumir de la siguiente manera. Como he mencionado anteriormente, cada vez que el usuario hace una búsqueda de tapas, las que selecciona se guardan en una propiedad llamada misTapas dentro de su nodo Person en la base de Neo4j. Pues bien, a partir de todas esas búsquedas, TapO'N sabe cuál es la tapa que más veces ha buscado el usuario y, como si de una búsqueda normal se tratase, pero en este caso sin que el usuario seleccione la tapa, cuando este pulsa el botón, se muestran por pantalla una serie de bares que ofrecen esa tapa más buscada (solo bares que no ha añadido a su lista, como sucede con las otras búsquedas).

```
Contador tapas: ▼ Object ⓘ
    Patatas fritas: 5
    Pollo: 4
```

Ilustración 63 Muestra por consola de cómo cuenta internamente las tapas la aplicación

Ahora explicaré de forma detallada cómo funciona cada parte, empezando por cómo se almacenan las búsquedas en el nodo Person.

```
buscarBares: function() {
  //Primero comprobamos que se ha seleccionado al menos una tapa
  if (this.tapasSeleccionadas.length == 0) {
    this.tipoAlerta = "error";
    this.alerta2 = true;
    this.textoAlerta = "Seleccione al menos una tapa";
  } else {
    //agregamos las tapas buscadas
    this.agregarMisTapas();
    console.log("Intento de buscar bares");
  }
}
```

Ilustración 64 - Muestra de la llamada a agregarMisTapas() cada vez que el usuario busca bares

Cuando el usuario accede a alguna de las búsquedas manuales, selecciona tapas y realiza la búsqueda, incluso aunque no encuentre bares, la aplicación guarda estas tapas seleccionadas y las agrega a su nodo.

En función de si es la primera vez que busca o no, la función llama al back con un valor que es verdadero o falso respectivamente. Esto es algo que afecta a cómo se añaden las tapas internamente en el backend, puesto que, si es la primera vez, en la base se tiene que crear la propiedad misTapas para el nodo y si ya hay otras, lo que hace es añadir las nuevas a la lista que ya tiene. Creo que es algo que no hace falta que detalle porque realmente lo que nos interesa aquí es lo que hace la aplicación una vez obtiene ese array de tapas del usuario.

```
agregarMisTapas: function() {
  //Si buscar Tapas devuelve falso, es porque aún no hay búsquedas
  //Por tanto, agregar será falso
  if (this.agregar == false) {
    console.log(
      "Intento de añadir las tapas por primera vez",
      this.tapasSeleccionadas
    );
    axios
      .post("http://localhost:3000/agregarMisTapas", {
        tapas: this.tapasSeleccionadas,
        user: this.idUsuario,
        nuevo: true
      })
      .then(response => {
        console.log("Datos recibidos: " + response.data.ok);
        //Llamada exitosa
        if (response.data.ok == true) {
          console.log("Se han añadido las tapas");
        } else {
          console.log("Fallo en la agregación de las tapas");
        }
      })
      .catch(error => {
        //Error al añadir el bar
        console.log(error);
      });
  }
}
```

Ilustración 65 - Muestra de cómo se agregan tapas por primera vez en el frontend

```
//Método que agrega una tapa a la lista de Mis Tapas
app.post('/agregarMisTapas', function (req, res) {
  // Devolver todos si se para un json vacio
  var tapas = req.body.tapas;
  var user = req.body.user;
  var nuevo = req.body.nuevo;
  console.log('Petición de añadir la tapa ', tapas)
  const sessionOtra = driver.session();
  if (tapas.length == 1) {
    console.log('Tapas 0: ' + tapas[0])
  }
  var query = "MATCH (n:Person) WHERE n.user='" + user + "' SET n.misTapas =";

  //Si aún no tiene tapas guardadas:
  if (nuevo == true) {
    query = query.concat("[")

    for (var i = 0; i < tapas.length; i++) {
      console.log('Tapas ' + i + " " + tapas[i])
      if (i == 0)
        query = query.concat("'" + tapas[i] + "'")
      else
        query = query.concat(", '" + tapas[i] + "'")
    }
    query = query.concat("]")
  }
  else if (nuevo == false) {
    query = query.concat("n.misTapas ")
    for (var i = 0; i < tapas.length; i++) {

```

Ilustración 66 - Función del backend que añade las tapas al nodo del usuario


```
{
  "identity": 2,
  "labels": [
    "Person"
  ],
  "properties": {
    "name": "Marta M",
    "password": "Marta123",
    "user": "Marta",
    "misTapas": [
      "Embutido",
      "Queso",
      "Embutido",
      "Caracoles",
      "Queso",
      "Nachos",
    ]
  }
}
```

Ilustración 67 - Nodo usuario en la base de Neo4j

Esto lo que hace es crear o ampliar la propiedad “misTapas” del nodo en cuestión, añadiendo una tras otra con cada una de sus búsquedas.

Una vez que sabemos cómo se añaden las tapas, nos interesa saber cómo se obtienen y no solo eso, cómo se procesan, porque necesitamos saber todas las tapas distintas y contar sus apariciones, de forma que podamos mostrar las tapas buscadas y usar ese número para el algoritmo de recomendación.

Para saber cómo se obtienen, tenemos que volver al punto que expliqué de Inicio, en el anterior apartado, donde existe una función, “cargarMisTapas()” que llama al back y obtiene la propiedad comentada del usuario en cuestión.

```
cargarMisTapas: function() {
  console.log("Intento de mostrar las tapas");
  axios
    .post("http://localhost:3000/misTapas", {
      user: this.idUsuario
    })
    .then(response => {
      console.log("Datos recibidos: " + response);
      //Llamada exitosa
      if (response.data.ok == true) {
        this.misTapas = response.data.datos;
        console.log(response.data + " Tapas recibidas");

        this.misTapas.forEach(el => {
          this.contador[el] = (this.contador[el] || 0) + 1;
        });
        console.log("Contador tapas: ", this.contador);
        for (var cont in this.contador) {
          this.tapasFiltro.push(cont);
          console.log("Tapa: ", cont);
        }
        console.log("El array de tapas filtrado: ", this.tapasFiltro);
      }
    })
}
```

Ilustración 68- Función que carga las tapas de un usuario en el frontend

```
this.misTapas.forEach(el => {
  this.contador[el] = (this.contador[el] || 0) + 1;
});
console.log("Contador tapas: ", this.contador);
for (var cont in this.contador) {
  this.tapasFiltro.push(cont);
  console.log("Tapa: ", cont);
}
```

Ilustración 69 - Bucles que cuentan y filtran las tapas

La clave de esta función está en este bucle que, a partir de ese array sin procesar, elabora una lista que contiene cada tapa y su número de apariciones. Una vez tenemos esto, es fácil formular una función que

permita recorrer esa lista y por comparación, coger la tapa que tiene un número mayor de apariciones.

```
recomendacion: function() {
  //Aquí usaremos el algoritmo de recomendación planteado
  //Tenemos un contador de cada aparición de cada tapa en la búsqueda
  var tapaMasBuscada = 0;
  var nombreTapa = "";
  for (var i = 0; i < this.tapasFiltro.length; i++) {
    if (this.contador[this.tapasFiltro[i]] > tapaMasBuscada) {
      tapaMasBuscada = this.contador[this.tapasFiltro[i]];
      nombreTapa = this.tapasFiltro[i];
    }
  }
  //Mostramos el valor en el log solo para dar valor a lo que hemos encontrado
  console.log(
    "La tapa más buscada es: " +
    nombreTapa +
    " con valor: " +
    tapaMasBuscada
  );
  if (this.tapasFiltro.length == 0) {
    //Ahora llamaremos al método de la búsqueda para esa tapa
    this.tipoAlerta = "error";
    this.alerta1 = true;
    this.textoAlerta1 = "¡Aún no hay búsquedas guardadas!";
  } else if (this.tapasFiltro.length != 0) {
    this.tipoAlerta = "success";
    this.alerta1 = true;
    this.textoAlerta1 = "¡Bares encontrados!";
    console.log("Intento de buscar bares");
    axios
      .post("http://localhost:3000/recomendacion", {
        tapa: nombreTapa,
        user: this.idUsuario
      })
      .then(response => {
        console.log("Datos recibidos: " + response.data.ok);
        //Llamada exitosa
        if (response.data.ok == true) {
          this.baresReco = response.data.datos;
          this.tipoAlerta = "success";
          this.alerta2 = true;
          this.textoAlerta1 = "¡Bares encontrados!";

          console.log("Se han encontrado los bares");
          setTimeout(() => {
            this.$router.push({
              name: "Bares",
              params: {
                idUsuario: this.idUsuario,
                bares: this.baresReco,
                voyATener: "SI"
              }
            });
          }, 750);
        }
      });
  }
}
```

Ilustración 70 - Función que realiza la recomendación en el frontend

Por tanto, lo que hace el front al pulsar el botón es el bucle de comparación que mencionaba. Si hubiera dos o más tapas con el mismo número de apariciones, usaría una de ellas para la búsqueda de los bares. Esta función tiene una serie de alertas que cubren los distintos casos posibles (si no hay búsquedas previas o no encuentra bares porque el usuario ha añadido todos en los que hay su tapa más buscada, que serían alertas de error).

6. Análisis de los resultados

En este apartado se muestran 2 ejemplos de funcionamiento de la aplicación.

- En primer lugar, el caso de un usuario que ha realizado muchas búsquedas y tiene bares guardados. Se muestra cómo el usuario puede ver sus bares y sus tapas, cómo puede borrarlos y cómo afecta esto en la base. También se muestra lo que le ofrece el algoritmo de recomendación.
- En segundo lugar, un usuario nuevo. Se muestra cómo realiza un par de búsquedas, una normal y otra personalizada, y estas se añaden en su nodo en la base, también añade bares y luego accede a la búsqueda recomendada.

➤ Primer caso:

Iniciamos sesión con Paco, un usuario ya veterano en la aplicación, que tiene muchas búsquedas realizadas y bares añadidos a su lista.



Ilustración 71 - Inicio de sesión con Paco

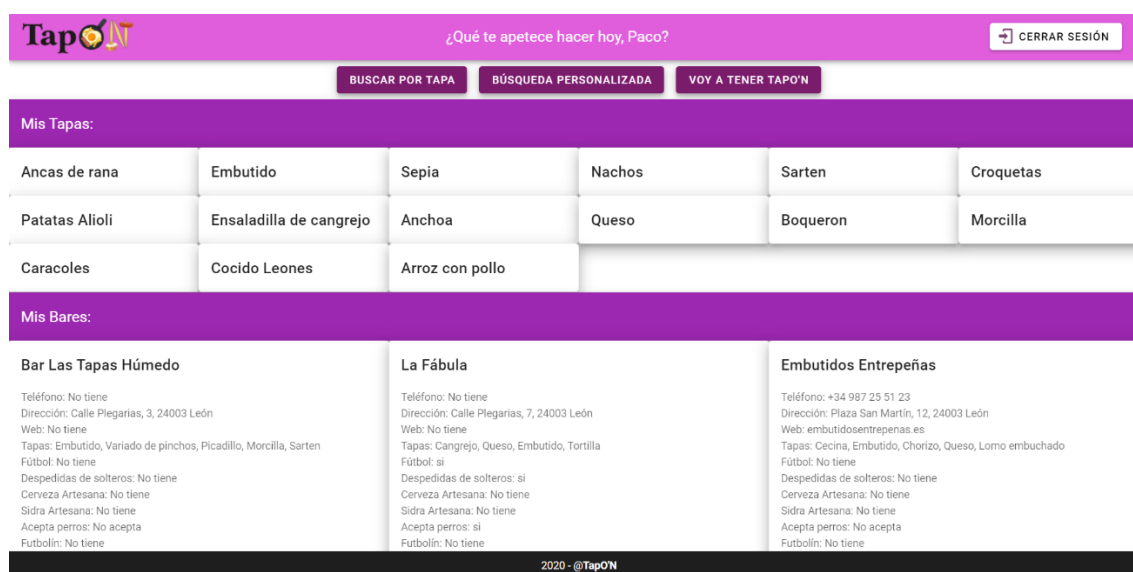


Ilustración 72 - Pantalla de Inicio de Paco

En la base podemos ver las propiedades de Paco y sus relaciones con bares:

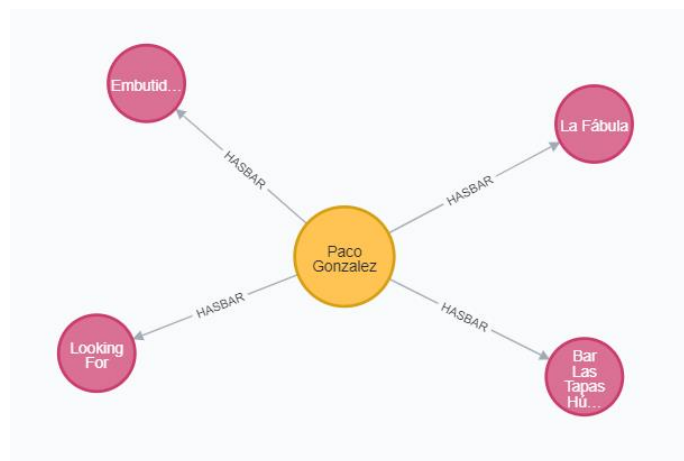



Ilustración 73 - Relaciones del usuario Paco con sus bares guardados

```
neo4j$ MATCH (n:Person)-[:HASBAR]-(b:Bar) WHERE n.user="Paco" RETURN n, b
```

n	b
<pre>{ "identity": 121, "labels": ["Person"], "properties": { "name": "Paco Gonzalez", "password": "Paco123", "user": "Paco", "misTapas": ["Ancas de rana", "Embutido", "Sepia", "Nachos", "Sarten", </pre>	<pre>{ "identity": 221, "labels": ["Bar"], "properties": { "name": "Bar Las Tapas Húmedo", "address": "Calle Plegarias, 3, 24003 León" } }</pre>

Ilustración 74 - Nodo Person de Paco y sus propiedades

Si borramos un par de bares y ejecutamos de nuevo la consulta, podemos ver que se han borrado:




¿Qué te apetece hacer hoy, Paco?

CERRAR SESIÓN

Ancas de rana	Embutido	Sepia	Nachos	Sarten	Croquetas
Patatas Alioli	Ensaladilla de cangrejo	Anchoa	Queso	Boqueron	Morcilla
Caracoles	Cocido Leones	Arroz con pollo			

Mis Bares:

 ¡Bar borrado correctamente!

Bar Las Tapas Húmedo

Teléfono: No tiene
Dirección: Calle Plegarias, 3, 24003 León
Web: No tiene
Tapas: Embutido, Variado de pinchos, Picadillo, Morcilla, Sarten
Fútbol: No tiene
Despedidas de solteros: No tiene
Cerveza Artesana: No tiene
Sidra Artesana: No tiene
Acepta perros: No acepta
Futbolín: No tiene

BORRAR BAR DE MIS BARES

Embutidos Entrepeñas

Teléfono: +34 987 25 51 23
Dirección: Plaza San Martín, 12, 24003 León
Web: embutidosentrepenas.es
Tapas: Cecina, Embutido, Chorizo, Queso, Lomo embuchado
Fútbol: No tiene
Despedidas de solteros: No tiene
Cerveza Artesana: No tiene
Sidra Artesana: No tiene
Acepta perros: No acepta
Futbolín: No tiene

BORRAR BAR DE MIS BARES

2020 - @TapON

Ilustración 75 - Paco borra un bar

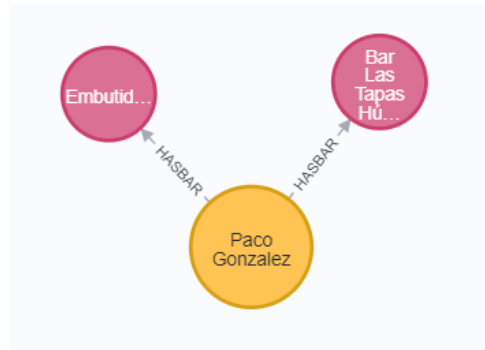


Ilustración 76 – Nodo de Paco una vez ha borrado 2 bares

Si queremos que la aplicación nos recomiende una tapa, podemos observar en consola cómo ha cargado las tapas al cargar la vista de Inicio y lleva un contador de las búsquedas de Paco (esto está explicado en el apartado correspondiente al algoritmo de recomendación).

```

Intento de inicio de sesión
true Inicio de sesión correcto
Intento de mostrar los bares
Intento de mostrar las tapas
Datos recibidos: [object Object]
[object Object] Tapas recibidas
Contador tapas:
▼ {Ancas de rana: 4, Embutido: 14, Sepia: 1, Nachos: 8, Sarten: 1, ...} ⓘ
  Ancas de rana: 4
  Anchoa: 9
  Arroz con pollo: 1
  Boqueron: 6
  Caracoles: 2
  Cocido Leones: 1
  Croquetas: 2
  Embutido: 14
  Ensaladilla de cangrejo: 1
  Morcilla: 1
  Nachos: 8
  Patatas Alioli: 1
  Queso: 11
  Sarten: 1
  Sepia: 1
  ▶ __ob__: Observer {value: {...}, dep: Dep, vmCount: 0}
  ▶ __proto__: Object
  
```

Ilustración 77 - Contador de las tapas buscadas por Paco

Podemos ver que la tapa que más veces ha buscado es Embutido con 14 apariciones. Si pulsamos en el botón de “Voy a tener TapO’N”, ejecuta el algoritmo y podemos ver cómo encuentra que esa tapa es la más buscada y con ella muestra los bares:

```

La tapa más buscada es: Embutido con valor: 14
Intento de buscar bares
  
```

Ilustración 78 - Llamada a la recomendación usando la tapa más buscada

Bares encontrados:

Mesón El Tizón
 Teléfono: +34 987 25 60 49
 Dirección: Calle Carnicerías, 1, 24003 León
 Web: restauranteeltizon.net
 Tapas: Embutido, Tortilla, Lomo embuchado, Chorizo, Gambas
 Fútbol: si
 Despedidas de solteros: No tiene
 Cerveza Artesana: No tiene
 Sidra Artesana: si
 Acepta perros: No acepta
 Fútbolín: No tiene

La Fábula
 Teléfono: No tiene
 Dirección: Calle Plegarias, 7, 24003 León
 Web: No tiene
 Tapas: Cangrejo, Queso, Embutido, Tortilla
 Fútbol: si
 Despedidas de solteros: si
 Cerveza Artesana: No tiene
 Sidra Artesana: No tiene
 Acepta perros: si
 Fútbolín: No tiene

Sidarta
 Teléfono: +34 987 04 04 53
 Dirección: Calle Cascalería, 3, 24003 León
 Web: gastrolateria-sidarta.negocio.site
 Tapas: Embutido, Nachos
 Fútbol: si
 Despedidas de solteros: No tiene
 Cerveza Artesana: si
 Sidra Artesana: No tiene
 Acepta perros: si
 Fútbolín: No tiene

AÑADIR A MIS BARES

AÑADIR A MIS BARES

AÑADIR A MIS BARES

Latino Bar
 Teléfono: +34 987 26 21 09
 Dirección: Plaza San Martín, 10, 24003 León
 Web: latinorestaurant.com
 Tapas: Oreja, Patatas Alioli, Embutido
 Fútbol: si

Taberna Oriente Medio
 Teléfono: +34 987 08 62 32
 Dirección: Calle Juan de Arfe, 8, 24003 León
 Web: No tiene
 Tapas: Caldo, Chorizo, Queso, Embutido
 Fútbol: No tiene


Ilustración 79 - Bares obtenidos tras ejecutar el algoritmo de recomendación

Se puede observar que en estos bares recomendados hay Embutido como tapa.

➤ Segundo caso:

Ahora vamos a crear un nuevo usuario, Rosa, que no va a tener ni tapas ni bares guardados por motivos obvios.

Registro



Nombre de Usuario

Rosa

Nombre Completo

Rosa García

Contraseña

.....

←


REGISTRARME

MATCH (n:Person) WHERE n.user="Rosa" RETURN n

n

```
{
  "identity": 3,
  "labels": [
    "Person"
  ],
  "properties": {
    "name": "Rosa García",
    "password": "Rosa123",
    "user": "Rosa"
  }
}
```

Ilustración 80 - Nodo de Rosa creado al hacer el registro



¿Qué te apetece hacer hoy, Rosa?

CERRAR SESIÓN

BUSCAR POR TAPA

BÚSQUEDA PERSONALIZADA

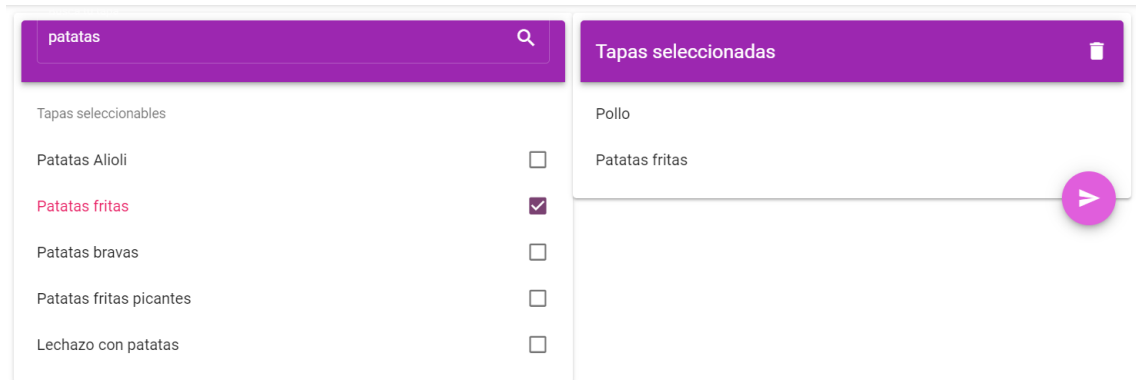
VOY A TENER TAPON

Mis Tapas:

Mis Bares:

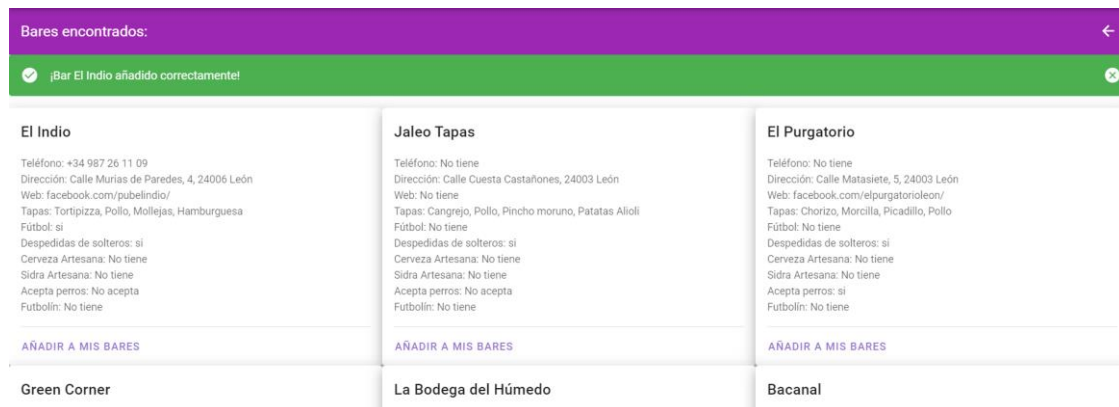
Ilustración 81 - Pantalla de Inicio de Rosa

Primero realizamos una búsqueda por tapa seleccionando Pollo y Patatas Fritas y añadimos un par de bares, después hacemos una personalizada con Pollo y Fútbol. Podemos ver que los bares tienen las características seleccionadas, aunque en la personalizada solo aparece un resultado porque solo tenemos un bar en la base así.



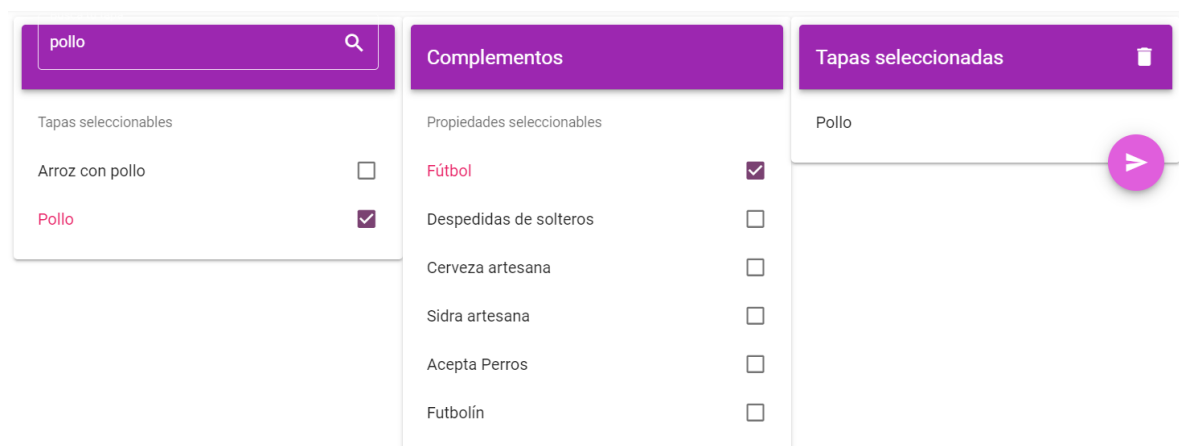
The interface shows a search bar with 'patatas' and a magnifying glass icon. Below it, a list of 'Tapas seleccionables' includes 'Patatas Alioli', 'Patatas fritas' (highlighted in red), 'Patatas bravas', 'Patatas fritas picantes', and 'Lechazo con patatas'. To the right, a 'Tapas seleccionadas' box contains 'Pollo' and 'Patatas fritas'. A green arrow button is at the bottom right.

Ilustración 82 - Primera búsqueda de tapas que realiza Rosa



The results are displayed in a grid. Each bar entry includes its name, contact information (phone, address, website), a list of tapas, and a 'Fútbol' status. Below each entry is a button labeled 'AÑADIR A MIS BARES'. A green notification bar at the top says '¡Bar El Indio añadido correctamente!'.

Ilustración 83 - Resultados obtenidos al buscar bares con Pollo y Patatas fritas



The interface shows a search bar with 'pollo' and a magnifying glass icon. Below it, a list of 'Tapas seleccionables' includes 'Arroz con pollo' and 'Pollo' (highlighted in red). To the right, a 'Complementos' box contains 'Fútbol' (highlighted in red), 'Despedidas de solteros', 'Cerveza artesana', 'Sidra artesana', 'Acepta Perros', and 'Fútbolín'. Further right, a 'Tapas seleccionadas' box contains 'Pollo'. A green arrow button is at the bottom right.

Ilustración 84 - Segunda búsqueda de Rosa, esta vez es personalizada

Bares encontrados:

La Bodega del Húmedo

Teléfono: +34 987 07 61 28
 Dirección: Calle Plegarias, 8, 24003 León
 Web: labodegadelhumedo.com
 Tapas: Croquetas, Oreja, Patatas bravas, Pollo
 Fútbol: si
 Despedidas de solteros: No tiene
 Cerveza Artesana: No tiene
 Sidra Artesana: No tiene
 Acepta perros: No acepta
 Fútbolín: No tiene

[AÑADIR A MIS BARES](#)

Ilustración 85 - Resultados para la búsqueda con Pollo y Fútbol

Ahora, si miramos de nuevo Inicio, podemos observar cómo han quedado registradas nuestras búsquedas.

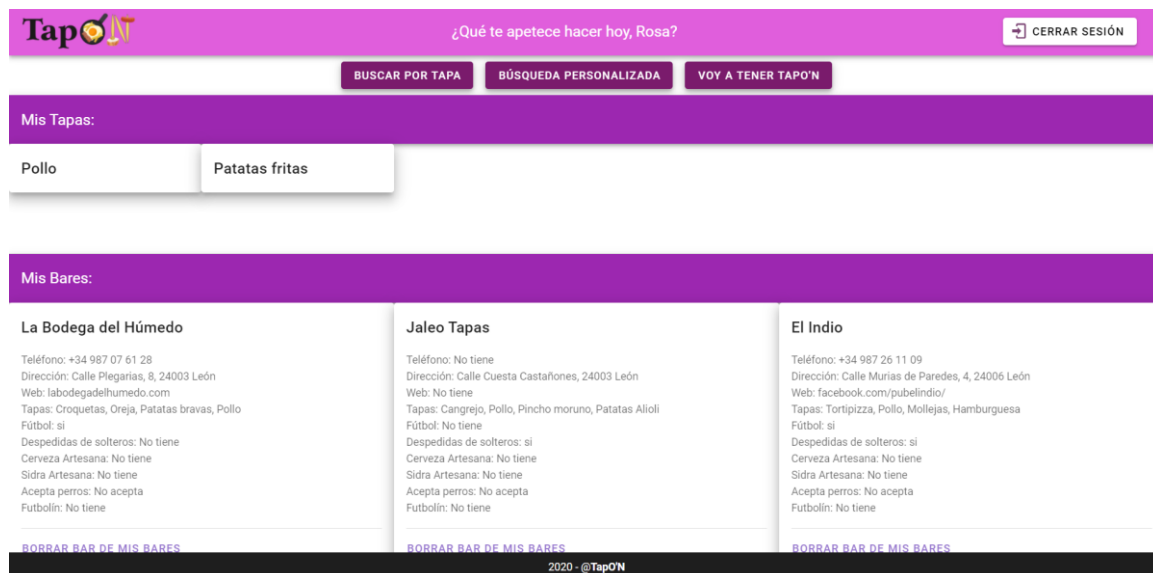


Ilustración 86 - Pantalla de Inicio de Rosa después de hacer las búsquedas

De la misma manera, el contador de las tapas que hemos buscado también corresponde y en la base se han guardado.

```
Datos recibidos: [object Object]
[object Object] Tapas recibidas
Contador tapas: ▼Object 3
  Patatas fritas: 1
  Pollo: 2
  __ob__: Observer {value: {...}, dep: Dep, vmCount: 0}
  __proto__: Object
Tapa: Pollo
Tapa: Patatas fritas
El array de tapas filtrado: ▶Array(2)
```

Ilustración 87 - Contador de las tapas buscadas por Rosa


```
MATCH (n:Person) WHERE n.user="Rosa" RETURN n
```

n

```
{
  "identity": 3,
  "labels": [
    "Person"
  ],
  "properties": {
    "name": "Rosa García",
    "password": "Rosa123",
    "user": "Rosa",
    "misTapas": [
      "Pollo",
      "Patatas fritas",
      "Pollo"
    ]
  }
}
```

Ilustración 88 - Nodo de Rosa en la base después de las búsquedas

Por lo tanto, si hacemos una recomendación, TapO'N nos mostrará bares donde haya Pollo y que no hayamos añadido a nuestra lista.

Bares encontrados:

El Purgatorio

Teléfono: No tiene
 Dirección: Calle Matasiete, 5, 24003 León
 Web: facebook.com/elpurgatorioleon/
 Tapas: Chorizo, Morcilla, Picadillo, **Pollo**
 Fútbol: No tiene
 Despedidas de solteros: si
 Cerveza Artesana: No tiene
 Sidra Artesana: No tiene
 Acepta perros: si
 Fútbolín: No tiene

[AÑADIR A MIS BARES](#)

Green Corner

Teléfono: +34 987 02 04 38
 Dirección: Calle Paloma, numero 1, 24003 León
 Web: No tiene
 Tapas: Sarten, Croquetas, Patatas Alioli, **Pollo**
 Fútbol: No tiene
 Despedidas de solteros: si
 Cerveza Artesana: No tiene
 Sidra Artesana: No tiene
 Acepta perros: si
 Fútbolín: No tiene

[AÑADIR A MIS BARES](#)

Ilustración 89 - Recomendación personalizada para Rosa

7. DAFO

En este apartado analizaré con detalle las Debilidades, Amenazas, Fortalezas y Oportunidades del desarrollo de TapO'N.

- **Debilidades:** con respecto a este punto creo que lo más destacable es la base de datos, que, aunque contiene 40 bares y sus tapas, es relativamente pequeña y en muchos casos ofrece una o dos opciones en cuanto a bares. Es evidente que, al hacerlo a mano, en el tiempo que tenía, no podía meter todos los que quisiera, así que puse la mayor cantidad que pude.
- **Amenazas:** Neo4j es la principal amenaza por motivos obvios, el desconocimiento que tenía de la herramienta y el hecho de que, aunque cada vez va habiendo más información, en muchos casos trabajé en el backend por intuición empleando mis conocimientos previos de hacer llamadas a otras bases de datos. También está el hecho de que nunca había trabajado solo haciendo todo el proceso, así que no tenía todos los conocimientos necesarios.
- **Fortalezas:** la principal fortaleza de esta aplicación pienso que es su originalidad, creo que es una idea distinta, que se podría escalar y emplear para hacer un mapa de todos los bares de España incluso y tiene muchas posibilidades de ampliación, algunas de las cuales explico en el apartado 8. Además, es muy fácil de usar bajo mi punto de vista, y cumple su propósito, aunque es evidente que todo se podría mejorar y por eso he dejado tantos comentarios indicando qué hace y cómo podría actuar de otra manera en ciertos puntos.
- **Oportunidades:** todo lo que he aprendido y las posibilidades de ampliar la aplicación, que me hacen pensar que tengo buenas ideas y me impulsan de cara a otros trabajos. Creo que eso es lo más importante de todo este proyecto.

8. Líneas de futuro

Tengo una serie de ideas que no he implementado en la aplicación, principalmente por falta de tiempo, pero que podrían añadirse. Yo mismo he pensado continuar y acabar con el proyecto porque me entusiasma la idea de hacer algo novedoso y realmente útil como creo que puede ser esta aplicación.

- Mi primera idea de mejora sería implementar un sistema de valoraciones. Creo que lo más sencillo sería valorar a los bares y no a las tapas, algo que había pensado pero que siento que no tiene mucho sentido según está diseñada la aplicación. El objetivo es llegar al mayor público posible y por eso las tapas no son súper específicas, para que el usuario pueda encontrar varios bares al buscar y no solo uno que ofrezca una tapa de autor muy concreta (que atención, se puede implementar y de hecho hay algunas más específicas que solo muestran un bar al buscar).

Por eso creo que implementar un sistema de valoración de bares sería muy útil y permitiría filtrar aún más, de forma que podríamos ordenar los bares por mejor valorados, por ejemplo, y mostrar las estrellas o puntos de cada uno en base a las valoraciones recibidas.

Lo más fácil sería implementar valoraciones solo mediante puntos, pero se podría ampliar a poner comentarios para mejorar aún más la experiencia.

- Mi segunda idea sería ampliar TapO'N y hacer que sea también una red social de tapas, algo que estuve a punto de implementar, pero no lo hice por tiempo y me gustaría haberlo hecho. Mi idea inicial no era hacer solo un sistema de recomendación "individual" si no que el usuario pudiese buscar a sus amigos (como si fuese Instagram, por ejemplo) y ver sus tapas o sus bares. A partir de eso las posibilidades de recomendación aumentarían porque, al igual que otras te recomiendan usuarios en función de las personas a las que sigues, TapO'N podría recomendar bares en función de los amigos del usuario. Podría ser un sistema que obligue a ambas partes a aceptar la "solicitud de amistad" o permitir el "seguimiento" aunque la otra parte no siga a la primera. Si hago una ampliación de sus funciones seguramente lo implementaré.
- Por último (y esta idea no la voy a desarrollar porque es sencilla y no quiero extenderme más), convertir la aplicación web en una aplicación para móviles sería fundamental para que pudiera ser un éxito.

9. Lecciones aprendidas

Los últimos dos meses que he pasado pensando, estudiando o desarrollando esta aplicación han sido muy intensos. Ha sido una experiencia completamente nueva porque hasta ahora mis trabajos universitarios de desarrollo de software habían sido cooperativos. Encontrarme en la situación de tener que hacer todo el proceso, pensar en la idea, desarrollarla y ser capaz de llevarla a cabo...

Pensar la idea fue complicado porque suelo convencerme de que no tengo buenas ideas y en este caso no la tenía, porque era muy amplia inicialmente, imposible de abarcar, así que tuve que detenerme y mirar de cerca las cosas para poder determinar qué podía hacer y qué me llamaba. Las tapas y los bares no son lo que más amo del mundo, pero *“lo que nos gusta no es nuestro camino, ni tampoco lo que no nos gusta. A veces el rumbo puede estar en lo que nos provoca indiferencia, en aquello que no nos apasiona ni aborrecemos.”* (Espinosa, 2011)

Me siento bien, orgulloso de mi trabajo, más allá de lo que hayan hecho mis compañeros, que sinceramente me importa poco más allá de lo que pueda aprender de los suyos. Más allá de la nota, TapO'N me ha enseñado quién soy ahora mismo y sobre todo quién puedo llegar a ser si me esfuerzo e intento conseguir mis propósitos.

Me he frustrado muchas veces a lo largo del trabajo, pero cada vez que “caía” o tenía dudas, investigaba, miraba cómo hacer determinada cosa y de una forma u otra lo conseguía. No me he detenido y he seguido adelante como fuera y prácticamente siempre he conseguido poner los elementos de la forma que había imaginado en mi cabeza.

Quitando el hecho de que me hubiera gustado hacer más, siempre se puede hacer más, *“El mundo nunca es suficiente”*, como esa película que me gusta tanto de 007, estoy muy contento. Ilusionado cuando trabajaba como hacía tiempo que no lo estaba por nada, enamorado casi porque me acostaba pensando en el trabajo, me levantaba con alguna idea, comía pensando en ello, no sé, todo suena muy poético pero el proyecto ha ocupado mi pensamiento y mi tiempo de una forma que nunca había experimentado.

¿Significa esto que he aprendido mucho? Sinceramente sí, porque solo por la ilusión y la satisfacción que me da haberlo hecho y el esfuerzo que le he dedicado, creo que hasta soy una persona diferente. Ahora afronto lo que me queda de carrera con la ambición de ser feliz algún día trabajando en algo como esto, ojalá que teniendo tantas ganas como las que he tenido haciéndolo.

10. Agradecimientos

En este punto debo hacer una breve mención a una serie de personas que quizás nunca lean este trabajo pero que me gustaría, incluso sin poner sus nombres, que quedasen reflejadas al igual que cuando un autor escribe sus agradecimientos.

- A mi hermana, por ayudarme a crear el nombre y aguantarme durante todo el proceso.
- A mis padres por ayudarme a elegir los colores.
- A Álex y a Raúl, mis socios en la aventura del desarrollo de software en la universidad. Sin todos los trabajos que hicimos juntos antes y durante no habría sido posible.
- Y por último a Andrea por su ilusión contagiosa que me animó a hacer la mejor aplicación posible.

11. Bibliografía y referencias

GitHub del proyecto:

<https://github.com/rsearb00/TapON-SIGBI2020>

Citas:

Espinosa, A. (2011). *Si tú me dices ven lo dejo todo... pero dime ven*. Barcelona: Penguin Random House Grupo Editorial, S. A. U.

Herramientas empleadas:

GitHub. (2020). Obtenido de GitHub: <https://guides.github.com/activities/hello-world/>

Neo4j. (2020). Obtenido de Neo4j: <https://neo4j.com/>

Node.js. (2020). Obtenido de Node.js: <https://github.com/nodejs/node>

OSF. (2020). Obtenido de OSF: <https://osf.io/?goodbye=true>

Visual Studio Code. (2020). Obtenido de Visual Studio Code: <https://code.visualstudio.com/docs>

Vue.js. (2020). Obtenido de Vue.js: <https://vuejs.org/v2/guide/>

Vuetify. (2020). Obtenido de Vuetify: <https://vuetifyjs.com/en/introduction/why-vuetify/#getting-started>

Imágenes externas:

Imagen de la división interna de una aplicación web: <https://www.suratica.es/que-es-el-backend/>

Enlaces de interés:

- (1) TapeaLeón, web con bares de León y sus tapas, web: <https://tapealeon.com/>
- (2) Ejemplo de un sistema de recomendación con libros, web: <https://edlearningblog.wordpress.com/2018/03/08/neo4j-sistema-de-recomendacion/>
- (3) Ejemplos de uso de Neo4j en GitHub: <https://github.com/neo4j-examples>