Centro Profesional Universidad Europea

UNIVERSIDAD EUROPEA

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

CICLO FORMATIVO DE GRADO SUPERIOR DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROYECTO FIN DE CICLO

ZampApp: Aplicación Matchmaking de Restaurantes

Alberto Barba, Álvaro Blázquez, Raúl Carrizo y Carlos García

CURSO 2023-24

TÍTULO: ZampApp: Aplicación Matchmaking de Restaurantes

AUTOR: Alberto Barba Soriano, Álvaro Blázquez Vallejo, Raúl Carrizo Martín y Carlos García Hernández

TUTOR DEL PROYECTO: Carlos José Elvira Gómez y Jorge Saldaña Alegre

FECHA DE LECTURA: 10 de junio de 2024

CALIFICACIÓN:

Fdo: Carlos José Elvira Gómez

Tutor/a del Proyecto

RESUMEN:

Cuando quedamos con amigos, o se organiza cualquier evento social, la decisión de dónde comer es siempre la más complicada. En este Trabajo de Fin de Ciclo (TFC) se explora la solución de dicho dilema mediante el desarrollo de una aplicación móvil: ZampApp.
La principal apuesta de la aplicación es el desarrollo de una interfaz que convierta la selección en una actividad más interactiva. Se mostrarán tarjetas con imágenes de los lugares y los platos que ofrecen, las cuales el usuario podrá seleccionar o descartar simplemente deslizando. A través de estas elecciones se le mostrará una recomendación de restaurante al usuario, así como el contacto con el lugar.
Para su desarrollo se utilizarán, como tecnologías principales: Flutter, framework de Google con que nos da la posibilidad de desarrollar la aplicación para distintas plataformas; y la API de Google Maps, de donde extraeremos la información de los restaurantes cercanos al usuario. ZampApp no solo facilita la decisión de dónde comer, sino que también ofrece una experiencia de usuario atractiva y personalizada, aprovechando tecnologías avanzadas para resolver un problema común en las reuniones sociales.

ABSTRACT:

When meeting with friends or organizing any social event, deciding where to eat is always the most challenging part. This Final Degree Project (FDP) explores the solution to this dilemma through the development of a mobile application: ZampApp.
The main feature of ZampApp is the development of an interface that makes the selection process more interactive. The application displays cards with images of restaurants and their dishes, which users can select or dismiss by simply swiping. Based on these choices, the user will receive a restaurant recommendation, along with the contact information for the establishment.
For its development, the main technologies used are Flutter, Google's framework that allows for cross-platform application development, and the Google Maps API, from which we will extract information about nearby restaurants.
ZampApp not only facilitates the decision of where to eat but also offers an attractive and personalized user experience, leveraging advanced technologies to solve a common problem in social gatherings.

AGRADECIMIENTOS

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

En la era digital actual, donde la variedad de opciones gastronómicas es abrumadora, surge la necesidad de simplificar y agilizar el proceso de elección de restaurantes. Es común encontrarnos con la dificultad de seleccionar un lugar que se adapte a nuestros gustos y preferencias entre la multitud de opciones disponibles. Para abordar este inconveniente, proponemos el desarrollo de una aplicación innovadora basada en un sistema de "matching" que revolucione la forma en que elegimos dónde comer.

1.1. . Objetivos

El objetivo principal de este proyecto es crear una aplicación móvil que facilite la búsqueda, selección y reserva de restaurantes de acuerdo con los gustos y preferencias individuales de cada usuario. Para lograr este propósito, nos planteamos los siguientes objetivos específicos:

- Diseñar una interfaz intuitiva y atractiva que permita a los usuarios navegar de manera sencilla y disfrutar de una experiencia interactiva al buscar restaurantes.
- Desarrollo del algoritmo de “match” para buscar restaurantes.

1.2. . Motivación

La motivación detrás de este proyecto surge de la necesidad de simplificar un proceso cotidiano que puede resultar tedioso y frustrante para muchas personas, las cuales pueden no disponer del tiempo suficiente para destinarlo en la elección de un restaurante, impactando de forma positiva en la vida diaria de las personas. A continuación, se listan las motivaciones que han impulsado la formulación de los objetivos del anterior punto:

- **Exploración de nuevas tecnologías:**
  - El primer objetivo, familiarizarnos con el lenguaje Dart y el framework Flutter, surge de la necesidad de adquirir competencias en tecnologías emergentes en el ámbito del desarrollo de aplicaciones móviles. Dart y Flutter representan una opción prometedora para la creación de interfaces de usuario rápidas y atractivas, lo que motiva su estudio y aplicación en este proyecto.

- **Satisfacer una demanda de información actualizada:**
  - La integración de la API de Google Maps para obtener información detallada y actualizada sobre los restaurantes cercanos responde a la creciente demanda de aplicaciones móviles que proporcionen datos relevantes y en tiempo real. La posibilidad de acceder a información precisa sobre establecimientos gastronómicos contribuye a mejorar la experiencia del usuario y facilita la toma de decisiones en cuanto a dónde comer.

- **Optimización de la experiencia del usuario:**

- El diseño de una interfaz intuitiva y atractiva busca ofrecer a los usuarios una experiencia de navegación sencilla y agradable al buscar restaurantes. La usabilidad y el diseño visual juegan un papel crucial en la aceptación y satisfacción del usuario, por lo que se prioriza la creación de una interfaz que favorezca la interacción y la comprensión de la información.

- **Innovación en el ámbito gastronómico:**
  - El desarrollo de la aplicación de "match" de restaurante representa una oportunidad para innovar en el sector gastronómico, al combinar tecnologías avanzadas con la experiencia culinaria. Esta funcionalidad ofrece a los usuarios una forma única y personalizada de descubrir nuevos restaurantes, basada en sus preferencias y gustos individuales.

En resumen, las motivaciones que subyacen a los objetivos abarcan desde la exploración tecnológica hasta la mejora de la experiencia del usuario y la innovación en el ámbito gastronómico. El presente trabajo aspira a contribuir al avance del conocimiento en el desarrollo de aplicaciones móviles, así como a ofrecer una herramienta útil y atractiva para los amantes de la buena comida, simplificando así un proceso cotidiano y mejorando la calidad de vida de las personas.

1.3. . Antecedentes

El presente Trabajo de Fin de Ciclo (TFC) se sitúa en el contexto de la investigación y desarrollo de aplicaciones móviles, específicamente en el ámbito de la búsqueda y recomendación de restaurantes. Para comprender mejor el entorno y la relevancia de este proyecto, es necesario examinar diversos antecedentes relacionados con las tecnologías empleadas, investigaciones previas y el estado actual del sector gastronómico y tecnológico.

Tecnologías y herramientas existentes:
En el campo del desarrollo de aplicaciones móviles, han surgido diversas herramientas y frameworks para simplificar el proceso de creación de interfaces de usuario y la gestión de datos. En particular, Flutter ha ganado popularidad debido a su capacidad para desarrollar aplicaciones multiplataforma con un alto rendimiento y una apariencia nativa. Estas tecnologías proporcionan un entorno propicio para la implementación de interfaces intuitivas y atractivas, fundamentales para la experiencia del usuario en aplicaciones móviles.

Estado actual del sector:
En la actualidad, el uso de aplicaciones móviles para la búsqueda y reserva de restaurantes ha experimentado un crecimiento significativo. Los usuarios buscan cada vez más herramientas que les permitan encontrar establecimientos gastronómicos cercanos, consultar información detallada y realizar reservas de forma rápida y sencilla. Si bien existen aplicaciones similares que ofrecen recomendaciones de restaurantes, muchas de ellas se basan en reseñas y opiniones de otros usuarios, lo que puede resultar subjetivo y poco confiable. Nuestra propuesta se diferencia al utilizar un enfoque más personalizado, basado en las preferencias individuales de cada usuario.

Contexto práctico:

En el contexto práctico, la aplicación propuesta busca simplificar el proceso de selección de restaurantes para los usuarios, ofreciendo una interfaz intuitiva y atractiva que les permita explorar opciones cercanas y recibir recomendaciones personalizadas. Además, la integración de la API de Google Maps nos permite acceder a una amplia base de datos de restaurantes, garantizando información actualizada y detallada sobre cada establecimiento.

En resumen, este proyecto se posiciona como una solución innovadora y eficaz para el problema común en la elección de restaurantes. Los antecedentes presentados en este apartado proporcionan una visión completa del contexto en el que se enmarca el TFC, destacando la importancia y la relevancia de abordar la problemática de la búsqueda y recomendación de restaurantes en el contexto actual de desarrollo de aplicaciones móviles.

2. DESARROLLO DE LA PRÁCTICA

En esta sección se examinarán las tecnologías fundamentales empleadas en el desarrollo de la aplicación. Cada una de estas tecnologías desempeña un papel vital en la concepción y funcionalidad del proyecto. A lo largo de esta sección, se proporciona un análisis comprensible de cada tecnología, dirigido a lectores no especializados en aspectos técnicos del desarrollo de aplicaciones móviles. Se explorarán el framework Flutter y el lenguaje de programación Dart, así como las herramientas de Firebase y la integración de la API de Google Maps. Además, se incluirán referencias bibliográficas relevantes para respaldar el entendimiento y contextualizar el uso de estas tecnologías en el contexto de este trabajo.

2.1. . Material

Flutter

Flutter es un framework de desarrollo de aplicaciones móviles multiplataforma creado por Google. Se basa en el lenguaje de programación Dart y se destaca por su capacidad para generar interfaces de usuario atractivas y fluidas. Utilizando un único código base, Flutter permite compilar aplicaciones nativas para diversas plataformas como Android, iOS y web, lo que facilita la creación de experiencias consistentes y de alto rendimiento en diferentes dispositivos.

Características principales de Flutter:

- **Widgets personalizables**: Flutter ofrece una amplia gama de widgets personalizables que permiten construir interfaces de usuario flexibles y dinámicas. Estos widgets pueden adaptarse fácilmente a diferentes tamaños de pantalla y orientaciones, lo que garantiza una experiencia de usuario consistente en todos los dispositivos.
- **Rendimiento de alta velocidad**: Flutter utiliza su propio motor de renderizado para lograr un rendimiento rápido y fluido, incluso en dispositivos con recursos limitados. La arquitectura de Flutter permite una actualización eficiente de la interfaz de usuario, lo que garantiza una respuesta rápida a las interacciones del usuario.
- **Hot Reload**: Una de las características más destacadas de Flutter es su función de hot reload, que permite realizar cambios en el código y ver los resultados de inmediato en el emulador o dispositivo físico. Esta capacidad de recarga en caliente agiliza significativamente el proceso de desarrollo y depuración, permitiendo a los desarrolladores iterar rápidamente sobre el diseño y la funcionalidad de la aplicación.
- **Desarrollo multiplataforma**: Con Flutter, es posible desarrollar una única base de código y compilarla para ejecutarse en diferentes plataformas. Esto simplifica el proceso de desarrollo y mantenimiento, ya que los desarrolladores pueden centrarse en la creación de una sola aplicación en lugar de desarrollar y mantener versiones separadas para cada plataforma.
- **Comunidad activa**: Flutter cuenta con una comunidad de desarrolladores activa que contribuye con bibliotecas, paquetes y recursos de aprendizaje. La documentación

oficial de Flutter es amplia y detallada, lo que facilita el aprendizaje y la resolución de problemas.

Dart

Dart es un lenguaje de programación desarrollado por Google, diseñado para ser utilizado en el desarrollo de aplicaciones móviles, web y servidores. Dart se destaca por su sintaxis clara y concisa, su enfoque en la programación orientada a objetos y su capacidad para la compilación tanto a código nativo como a JavaScript. Utilizado como lenguaje principal en el desarrollo de aplicaciones con Flutter, Dart ofrece una serie de características que lo hacen adecuado para el desarrollo de aplicaciones modernas y escalables.

Características principales de Dart:

- **Sintaxis clara y concisa:** Dart utiliza una sintaxis familiar y fácil de entender, lo que facilita su aprendizaje y uso para desarrolladores de diferentes niveles de experiencia. La sintaxis de Dart se asemeja a la de otros lenguajes de programación populares como Java y JavaScript, lo que facilita la transición para aquellos que ya están familiarizados con esos lenguajes.
- **Programación orientada a objetos:** Dart es un lenguaje orientado a objetos en el que todo es un objeto. Esto permite a los desarrolladores estructurar su código de manera modular y reutilizable, lo que facilita la creación y mantenimiento de aplicaciones complejas.
- **Tipado estático opcional:** Dart admite el tipado estático opcional, lo que significa que los desarrolladores pueden optar por especificar tipos de datos para variables y parámetros de función si lo desean. Esto puede ayudar a detectar errores en tiempo de compilación y mejorar la legibilidad del código.
- **Compilación eficiente:** Dart se compila tanto a código nativo como a JavaScript, lo que permite ejecutar aplicaciones Dart en una amplia gama de plataformas, incluyendo dispositivos móviles, navegadores web y servidores. La compilación Just-In-Time (JIT) de Dart permite la ejecución y depuración rápida durante el desarrollo, mientras que la compilación Ahead-Of-Time (AOT) produce un código altamente optimizado para un rendimiento máximo en producción.
- **Soporte para asincronía:** Dart ofrece un conjunto de características para trabajar de manera eficiente con operaciones asíncronas, como Future y Stream, que permiten realizar operaciones de entrada/salida (E/S) sin bloquear el hilo principal de ejecución.

Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles y web creada por Google que proporciona una amplia gama de servicios en la nube para simplificar y acelerar el desarrollo de aplicaciones. En este proyecto, se emplean dos servicios clave de Firebase: Firestore y Authentication, que desempeñan roles fundamentales en el almacenamiento de datos y la gestión de usuarios, respectivamente.

Firestore es una base de datos en tiempo real y orientada a documentos. Permite almacenar y sincronizar datos en la nube de forma eficiente, lo que facilita la creación de aplicaciones escalables y colaborativas. Sus características principales incluyen:

- **Base de datos NoSQL:** Firestore utiliza una estructura de base de datos NoSQL flexible y escalable, basada en documentos y colecciones, lo que simplifica el modelado y la organización de datos.
- **Sincronización en tiempo real:** Los cambios realizados en los datos se reflejan instantáneamente en todos los dispositivos conectados, lo que garantiza una experiencia de usuario actualizada y consistente en tiempo real.
- **Capacidades de consulta avanzadas:** Firestore ofrece potentes capacidades de consulta que permiten filtrar, ordenar y limitar los datos de manera eficiente, lo que facilita la recuperación de la información necesaria para la aplicación.

Firebase Authentication proporciona servicios de autenticación segura y fácil de usar, lo que permite a los desarrolladores agregar fácilmente la autenticación de usuarios a sus aplicaciones. Sus características principales incluyen:

- **Diversos métodos de inicio de sesión**: Firebase Authentication admite múltiples métodos de inicio de sesión, incluidos correo electrónico y contraseña, autenticación mediante redes sociales como Google y Facebook, así como autenticación anónima.
- **Gestión de usuarios:** El servicio ofrece herramientas para la gestión completa de usuarios, que incluyen la creación, eliminación y actualización de cuentas de usuario, así como la verificación de correos electrónicos y la recuperación de contraseñas.
- **Seguridad integrada:** Firebase Authentication proporciona medidas de seguridad robustas, como la verificación de correo electrónico y el uso de tokens de acceso, para proteger las cuentas de usuario contra posibles ataques.

Google Maps API

Google Maps API es una poderosa herramienta para la integración de mapas y datos de ubicación en aplicaciones móviles y web. En este proyecto, Google Maps API se utiliza para obtener información detallada sobre los restaurantes cercanos, incluyendo su ubicación, etiquetas, horarios de funcionamiento y fotos. La aplicación permite a los usuarios deslizar imágenes de los restaurantes para indicar si les interesa o no, lo que proporciona una forma interactiva y visualmente atractiva de explorar los restaurantes de los alrededores.

Funcionalidades principales de Google Maps API:

- **Obtención de datos de ubicación:** Google Maps API proporciona acceso a una amplia base de datos de lugares, incluyendo restaurantes, tiendas y puntos de interés. Esto permite a la aplicación mostrar una lista de restaurantes cercanos y sus ubicaciones en un mapa interactivo.

- **Información detallada de los lugares:** Además de la ubicación, Google Maps API ofrece información detallada sobre cada lugar, como su nombre, dirección, horarios de funcionamiento, valoraciones y fotos. Esto permite a los usuarios obtener una vista completa de cada restaurante antes de tomar una decisión.
- **Integración de imágenes de restaurantes:** La aplicación utiliza la API de Google Maps para mostrar imágenes de los restaurantes de los alrededores, permitiendo a los usuarios deslizar las imágenes horizontalmente para indicar si les gusta o no cada restaurante. Esta funcionalidad proporciona una forma intuitiva y visualmente atractiva de explorar y evaluar los restaurantes cercanos.
- **Geolocalización del usuario:** Google Maps API permite la geolocalización del usuario, lo que permite a la aplicación mostrar la ubicación actual del usuario en el mapa y proporcionar recomendaciones de restaurantes basadas en su ubicación.

2.2. . Planificación

El éxito de cualquier proyecto depende en gran medida de una planificación efectiva que garantice la organización, la comunicación fluida y la asignación adecuada de tareas. En el marco de este Trabajo de Fin de Ciclo (TFC), se establecerá un plan detallado que abarque desde la distribución de responsabilidades hasta la definición de los pasos a seguir para alcanzar los objetivos propuestos.

Organización del equipo:
El equipo estará compuesto por Alberto, Álvaro, Raúl y Carlos; cada uno con roles y responsabilidades específicas en el desarrollo del proyecto. Se designará un líder de proyecto encargado de coordinar las actividades y garantizar el cumplimiento de los plazos establecidos.

Formas de comunicación:
Se establecerán diversas formas de comunicación para mantener una interacción constante entre los miembros del equipo. Se programarán reuniones regulares, preferiblemente semanales, para revisar el progreso, discutir posibles problemas y tomar decisiones importantes. Además, se utilizará WhatsApp para la comunicación diaria y la resolución de dudas o consultas urgentes.

Tareas de cada uno:
Las tareas se distribuirán equitativamente entre los miembros del equipo de acuerdo con sus habilidades y áreas de expertas. Por ejemplo, Alberto estará a cargo del desarrollo del frontend utilizando Flutter, mientras que Raúl se encargará de la integración de la API de Google Maps y la gestión de la base de datos. Se documentará claramente la asignación de tareas para evitar confusiones y solapamientos.

Objetivos a conseguir:
Los objetivos principales del proyecto incluyen la familiarización con el lenguaje Dart y el framework Flutter, la integración exitosa de la API de Google Maps, el diseño de una interfaz intuitiva y atractiva, y el desarrollo de la funcionalidad de "match" de restaurantes. Se espera alcanzar estos objetivos dentro de los plazos establecidos y cumplir con los estándares de calidad definidos.

Pasos a seguir:
Para alcanzar los objetivos propuestos, se seguirá un plan de trabajo estructurado en etapas definidas. Estas etapas incluirán la investigación y aprendizaje inicial sobre Dart y Flutter, la implementación de la funcionalidad de búsqueda de restaurantes utilizando la API de Google Maps, el diseño y desarrollo de la interfaz de usuario, y la implementación final de la funcionalidad de "match" de restaurantes. Cada etapa se completará según un calendario predefinido, con revisiones periódicas para evaluar el progreso y realizar ajustes si es necesario.

En resumen, la planificación detallada del proyecto garantizará una ejecución eficiente y exitosa, maximizando el aprovechamiento de los recursos disponibles y asegurando la consecución de los objetivos planteados en el tiempo previsto.

2.3. . Descripción del trabajo realizado

En este apartado se detallan los pasos realizados para alcanzar los objetivos del proyecto, incluyendo la planificación del desarrollo, el diseño de la interfaz, así como el desarrollo.

Planificación del desarrollo

Backlog del proyecto
El backlog del proyecto es una lista exhaustiva de todas las tareas, funcionalidades y mejoras planificadas para la aplicación de matchmaking de restaurantes. Este backlog se desarrolló y gestionó de manera colaborativa, incorporando constantemente los comentarios de los interesados y el equipo de desarrollo. A continuación, se detallan las categorías y elementos principales del backlog:

- **Funcionalidades principales:**
  - Registro de usuarios: Implementación de un sistema de registro y de inicio de sesión
  - Perfil de usuario: Desarrollo de perfiles de usuario, que incluya información básica, preferencias gastronómicas y otros detalles relevantes.
  - Matchmaking con restaurantes: Desarrollo del algoritmo de matchmaking que recomienda restaurantes basándonos en los “likes”
- **Mejoras de la experiencia de usuario:**
  - Interfaz de usuario intuitiva: Diseño y desarrollo de una interfaz de usuario amigable y fácil de usar para garantizar una experiencia fluida para los usuarios.
- **Integraciones externas:**
  - Integración de la API de Google Maps: Implementar las funcionalidades de geolocalización que ofrece Google, así como las peticiones necesarias a la API para obtener los datos de los restaurantes cercanos.
  - Integración de Firebase: Implementación de los servicios de Authentication y base de datos que ofrece Firebase.

Plan de desarrollo

El desarrollo de la aplicación se dividirá en tres sprints, en los que se perseguirá obtener un producto viable que se ajuste con los objetivos de cada sprint. A continuación, se detallan los principales aspectos de cada sprint:

Sprint 1: Preparación del Terreno y Diseño de la Interfaz de Usuario

- **Objetivos:**
  - Configurar el entorno de desarrollo y establecer la estructura del proyecto
  - Diseñar la interfaz de usuario, desarrollo del registro e inicio de sesión y prototipo del deslizamiento de tarjetas.
- **Backlog:**
  - Configuración del entorno de desarrollo.
  - Creación de los wireframes y mockups de la interfaz de usuario.
  - Implementación del sistema básico de registro y autenticación de usuarios.

Sprint 2: Desarrollo del sistema de Matchmaking y perfiles

- **Objetivos:**
  - Implementar el algoritmo de matchmaking que recomienda restaurantes a partir de la pantalla de deslizamiento de tarjetas
  - Implementación de la creación de perfiles al registrarse y visualización del perfil del usuario.
- **Backlog:**
  - Integración de la API de Google Maps
  - Desarrollo del algoritmo de matchmaking
  - Diseño e implantación de la interfaz de registro para crear un perfil del usuario.

Sprint 3: Revisión y pruebas

- **Objetivos:**
  - Revisar del trabajo realizado hasta ahora y acabar posibles tareas pendientes
  - Realización de las pruebas finales para asegurar el correcto funcionamiento de la aplicación
- **Backlog:**
  - Realización de las pruebas unitarias para verificar los resultados esperados
  - Resolver cualquier incidencia restante

Diseño

Log in:

En la página de inicio de sesión el usuario puede incluir sus credenciales. En caso de ser las correctas, se le redirigirá a la pantalla de selección de tarjetas.

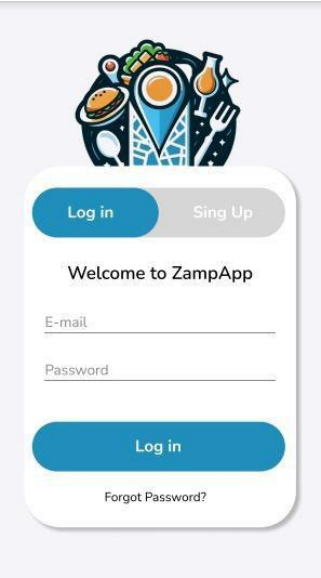


Figura 1. Diseño en Figma de la pantalla de inicio de sesión

Al pulsar en el botón de “Sign up”, el usuario será redirigido a la pantalla de registro.

Registro:

En la pantalla de registro se le pedirá al usuario el correo electrónico y una contraseña. Al darle a “Sign up”, en caso de que todo sea correcto, se registrará al usuario. La idea es dirigir al usuario por una serie de pantallas en las que se crea un perfil al usuario, que después se irá perfilando con el uso

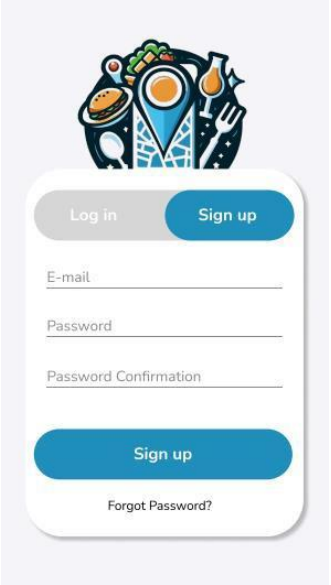


Figura 2. Diseño en Figma de la pantalla de registro de usuario.

Pantalla de Swipe:

Esta pantalla presenta una interfaz de swipe que muestra tarjetas de diferentes restaurantes. Cada tarjeta representa un restaurante distinto y contiene una imagen del lugar.

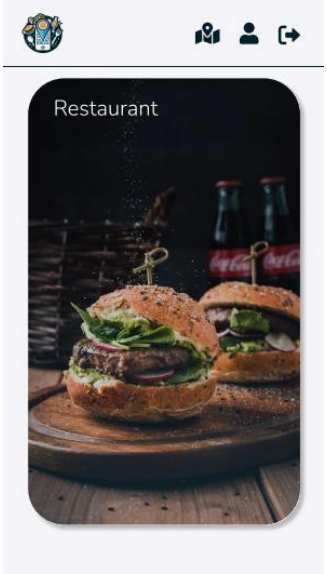


Figura 3. Diseño en Figma de la pantalla de Swipe.

En la barra superior encontramos tres opciones distintas. Al pulsar en el icono del mapa, el usuario verá un mapa de la zona del alrededores donde podrá configurar el rango de búsqueda de restaurantes; al pulsar en el icono de perfil de usuario, se dirige al usuario a la pantalla con la información de su perfil; por último, con el último icono el usuario cierra sesión.

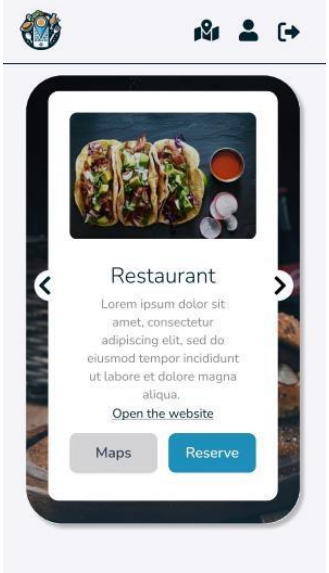


Figura 4. Diseño en Figma del cuadro de diálogo que aparece al hacer "match".

Cuando el algoritmo considera que se ha realizado un match, aparecerá una ventana informando al usuario, en la que podrá contactar con el restaurante o ver la ubicación de este.

Mapa:

En la pantalla del mapa, el usuario podrá utilizar el deslizador de la parte inferior para seleccionar el rango de búsqueda. Par confirmar el nuevo rango se deberá pulsar el botón "Guardar" ("Save").

Figura 5. Diseño en Figma de la pantalla de configuración del radio de búsqueda.

Perfil:

Esta pantalla muestra la información personal y de la cuenta del usuario, incluyendo detalles como la foto de perfil, nombre, dirección de correo electrónico y opciones para gestionar la contraseña.

Figura 6. Diseño en Figma de la pantalla de perfil del usuario.

- Elementos Destacados:
  - Foto de Perfil: Muestra la imagen de perfil del usuario, que puede ser personalizada o predeterminada.

- Información Personal: Muestra el nombre del usuario y su dirección de correo electrónico asociada a la cuenta.
- Gestión de Contraseña: Permite al usuario cambiar o actualizar la contraseña de su cuenta.

- Funcionalidad: El usuario puede ver y editar su información personal, incluida la foto de perfil y el nombre, puede cambiar la contraseña de su cuenta utilizando la opción de gestión de contraseña.

En función de cómo vaya el desarrollo, se plantea introducir una opción para ver estadísticas del usuario, por ejemplo: número de "matches", restaurantes preferidos, etc.

Desarrollo de la aplicación

El objetivo principal de este punto es darle funcionalidad a la interfaz que se ha diseñado en el punto anterior. Para ello se realizan las siguientes actividades:

- Configuración del entorno de desarrollo con Flutter.
- Desarrollo de la funcionalidad de selección mediante tarjetas.
- Integración de Firebase para la creación de cuentas de usuario.
- Implementación de la API de Google Maps.
- Desarrollo del algoritmo de "matches".

Configuración del entorno de desarrollo

El primer paso para poder empezar a desarrollar la aplicación es la preparación del entorno de desarrollo que, en nuestro caso, se va a realizar con Flutter.

Lo primero será elegir un editor de texto o IDE con el que trabajar, en nuestro caso la elección ha sido Visual Studio Code, ya que nos ofrece una gran capacidad de personalización para trabajar tal y como queremos, además de contar con la extensión oficial de Flutter, lo que lo convierte en una gran elección. Otras posibles opciones eran utilizar Android Studio o IntelliJ.

Es importante comentar que, para cualquiera de las elecciones de IDE, deberemos instalar previamente Git y Android Studio, este último es importante para poder descargar el SDK de Android, así como el emulador de Android.

Una vez elegido el IDE, deberemos instalar el SDK de Flutter. En cualquiera de las opciones que se presentan, contamos con una extensión o plugin que hace este paso mucho más sencillo. En el caso de VS Code, una vez tenemos la extensión, solo hay que escribir el comando 'flutter' en la paleta de comandos y seleccionar la opción de crear un nuevo proyecto. En caso de tener el SDK instalado, se ejecutará el comando; en caso contrario, podremos descargar el SDK desde el propio VS Code y añadirlo a nuestras variables de entorno.

Para probar que se ha instalado correctamente, basta con escribir 'flutter' en la terminal.

Figura 7. Ejecución del comando 'flutter'

Para poder desarrollar en Android deberemos tener el SDK de Android y el emulador de este, pero estos se instalan como parte de la instalación de Android Studio.

Por último, deberemos ejecutar el comando 'flutter doctor -v' para solucionar cualquier problema que haya podido aparecer o, como nos ha ocurrido, configurar correctamente la ruta del SDK de Android.

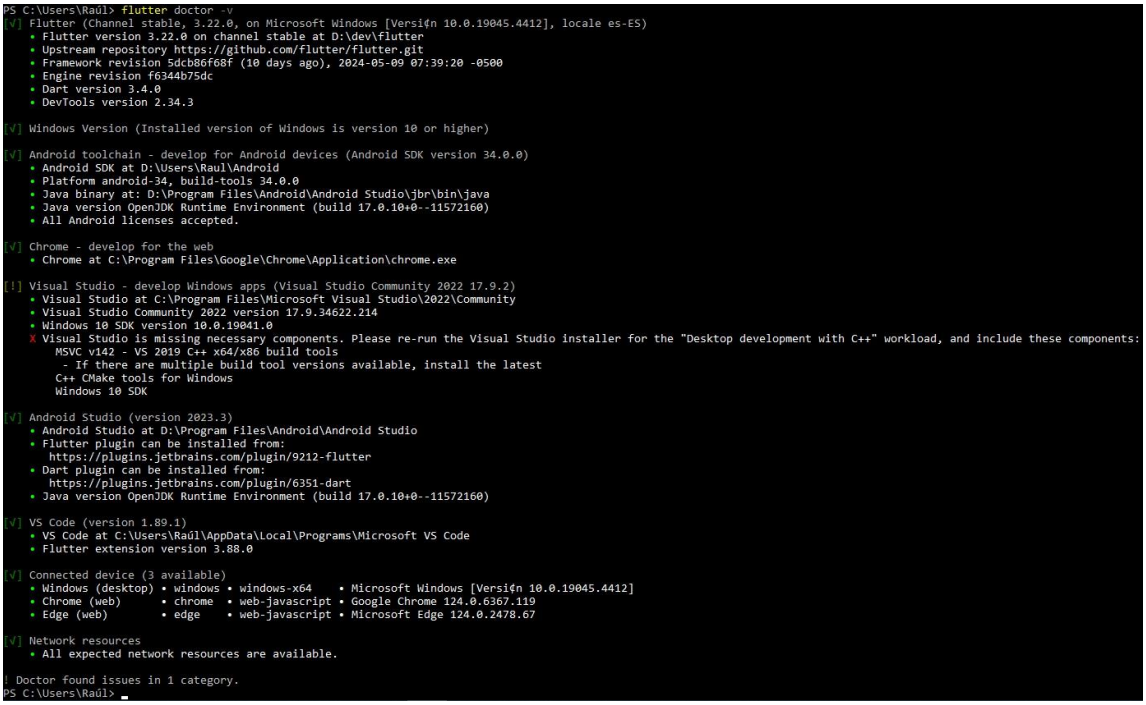
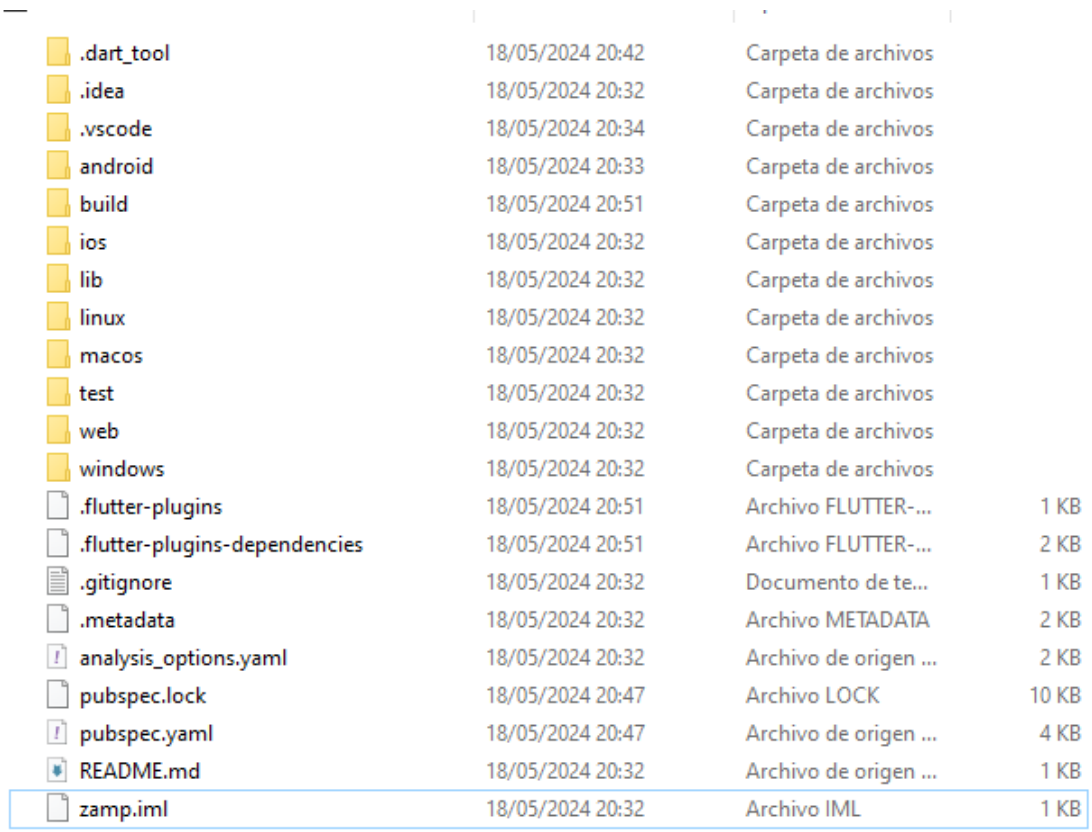


Figura 8. Ejecución de 'flutter doctor -v' sin errores.

Una vez tenemos configurado el entorno de desarrollo, podemos crear un nuevo proyecto de Flutter, desde VS Code como desde la propia terminal. Se nos creará un repositorio Git de forma automática, así como la estructura de directorios del proyecto para las distintas plataformas, además del de trabajo: 'lib'.



.dart_tool	18/05/2024 20:42	Carpeta de archivos	
.idea	18/05/2024 20:32	Carpeta de archivos	
.vscode	18/05/2024 20:34	Carpeta de archivos	
android	18/05/2024 20:33	Carpeta de archivos	
build	18/05/2024 20:51	Carpeta de archivos	
ios	18/05/2024 20:32	Carpeta de archivos	
lib	18/05/2024 20:32	Carpeta de archivos	
linux	18/05/2024 20:32	Carpeta de archivos	
macos	18/05/2024 20:32	Carpeta de archivos	
test	18/05/2024 20:32	Carpeta de archivos	
web	18/05/2024 20:32	Carpeta de archivos	
windows	18/05/2024 20:32	Carpeta de archivos	
.flutter-plugins	18/05/2024 20:51	Archivo FLUTTER-...	1 KB
.flutter-plugins-dependencies	18/05/2024 20:51	Archivo FLUTTER-...	2 KB
.gitignore	18/05/2024 20:32	Documento de te...	1 KB
.metadata	18/05/2024 20:32	Archivo METADATA	2 KB
analysis_options.yaml	18/05/2024 20:32	Archivo de origen ...	2 KB
pubspec.lock	18/05/2024 20:47	Archivo LOCK	10 KB
pubspec.yaml	18/05/2024 20:47	Archivo de origen ...	4 KB
README.md	18/05/2024 20:32	Archivo de origen ...	1 KB
zamp.iml	18/05/2024 20:32	Archivo IML	1 KB

Figura 9. Estructura de proyecto en Flutter.

Desarrollo de la funcionalidad de selección de tarjetas.

Lo primero que se desarrolló fue la funcionalidad principal y sobre lo que se fundamenta la aplicación: la selección mediante tarjetas.
Gracias a trabajar con Flutter esta parte se simplificó considerablemente, ya que tiene integrados una gran cantidad de Widgets que sirven como plantilla para desarrollar prácticamente lo que se te ocurra. Esta pantalla está basada en el uso de tres widgets: PageView, Dismissible y Card.

El widget Card se encarga de envolver otro en una tarjeta tal y como su nombre indica. Esta tarjeta se puede modificar a nuestro gusto para darle el aspecto que deseamos.

Dismissible envuelve el widget de Card que contendrá la imagen. Este widget crea un componente el cual podemos descartar simplemente arrastrando. Cuando se descarta, lanza un evento del cual podemos extraer la dirección en la que se ha descartado, pudiendo determinar si debemos tratarlo como un “like” o no.

```dart
return Dismissible(
  key: Key(
      originalCards[index]), // Clave única para cada tarjeta // Key
  direction: DismissDirection
      .horizontal, // Solo permite deslizar horizontalmente
  onDismissed: (direction) {
    setState(() {
      var currentCard = originalCards.removeAt(index);
      if (direction == DismissDirection.startToEnd) {
        selectedCards.add(
            currentCard); // Si se desliza hacia la derecha, añade la carta a las seleccionadas
      }
    });
  },
```

Figura 10. Código de widget Dismissible.

Como se puede ver en la Figura 10, podemos limitar el deslizamiento a que solo pueda ser horizontal y, cuando se descarta, borramos la carta de la lista donde están guardadas e introducirla en las seleccionadas.

Finalmente, envolvemos el Dismissible en un widget PageView con el constructor builder() para asegurarnos que cada vez que se deslice una carta, aparezca la siguiente.

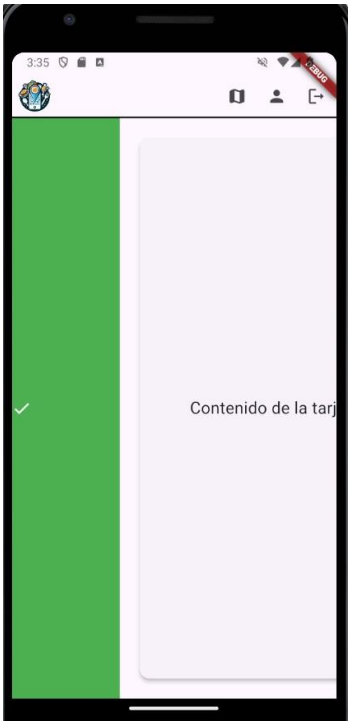


Figura 11. Prototipo de la pantalla de selección de tarjetas.

Integración de Firebase para la creación de cuentas de usuario

El siguiente paso fue conectar nuestra aplicación con Firebase, ya que queríamos añadir cuanto antes la posibilidad de crear cuentas y empezar a desarrollar la navegabilidad. Optamos por un

inicio de sesión y registro simples, basándonos únicamente en el correo electrónico y la contraseña.

Para conectar Firebase lo primero es descargar e instalar la CLI (Command Line Tools). Una vez instalada, iniciamos sesión en nuestra cuenta de Firebase desde esta. Añadimos las dependencias de Firebase en el archivo 'pubspec.yaml' y ejecutamos 'flutterfire configure' para terminar de conectar la aplicación. A partir de aquí, solo debemos llamar a los métodos 'createUserWithEmailAndPassword' y 'signInWithEmailAndPassword' para crear las cuentas e iniciar sesión respectivamente.

La funcionalidad que hemos añadido para evitar que el usuario tenga que estar iniciando sesión cada vez que abra la aplicación, además de mostrar una pantalla de inicio distinta a nuevos usuarios, es la adición de la biblioteca "SharedPreferences" de Flutter que permite guardar información simple de forma local en el dispositivo. Combinada con el atributo 'currentUser' de la instancia de Firebase, podemos hacer que la experiencia de nuevos usuarios y los ya registrados sea más agradable.

```
Future<void> checkFirstTime() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();

  isFirstTime = prefs.getBool('isFirstTime') ?? true;
  print(isFirstTime);
  if (!isFirstTime) {
    User? user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      Navigator.of(context).pushReplacement(
          MaterialPageRoute(builder: (context) => CardSelector()));
    } else {
      Navigator.of(context)
          .pushReplacement(MaterialPageRoute(builder: (context) => LoginScreen()));
    }
  } else {
    await prefs.setBool('isFirstTime', false);

    Navigator.of(context)
        .pushReplacement(MaterialPageRoute(builder: (context) => RegisterScreen()));
  }
}
```

Figura 12. Lógica del primer inicio y mantener la sesión iniciada.

Implementación de la API de Google Maps

El siguiente paso del desarrollo fue la implementación de las librerías necesarias para obtener la ubicación actual del usuario, el mapa que muestra el radio en el que se están buscando restaurantes y la llamada a la API de Google Maps para obtener la información de los restaurantes cercanos.

Para obtener la ubicación del usuario, se hace uso de la librería 'geolocator' de Flutter cuyo único requerimiento es añadirla a las dependencias del proyecto, además de actualizar el 'AndroidManifest.xml' para indicar los permisos necesarios.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

Figura 13. Permisos para obtener ubicación en Android.

La primera vez que se abre la aplicación ejecutamos los métodos encargados de geolocalizar el dispositivo y guardamos la localización a nivel de la aplicación para que sea fácilmente accesible desde cualquier pantalla de esta. Esto lo logramos con la clase propia Location que extiende de ChangeNotifier, que se instanciará al iniciar y, cualquier cambio de valor de sus atributos, se notificarán a todas las pantallas que en ese momento estén activas.

Antes de poder empezar a hacer llamadas a Google Places (API con los detalles de los lugares), necesitamos que el usuario sea capaz de configurar el radio de búsqueda . Para ello, desarrollamos una pantalla que muestra un mapa con la ubicación actual, además de un "slider" para modificar el radio de búsqueda cómodamente. El radio se muestra de forma visual en el mapa y se guarda automáticamente en Location, para que se pueda acceder desde la pantalla de selección de tarjetas, que es la que se encarga de realizar la llamada a la API.

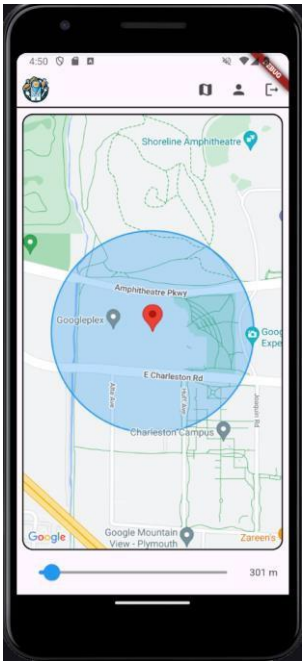


Figura 14. Implementación de la pantalla del mapa.

Para hacer llamadas a la API, nos creamos una clase que contiene los métodos que realizarán las llamadas. Estos métodos trabajarán con objetos Restaurante (Figura 15) que nos facilitarán tratar con la información que recibimos.

```
Restaurante({
  required this.name,
  required this.address,
  this.isOpen,
  this.website,
  this.phoneNumber,
  required this.photos,
  required this.tags,
});
```

Figura 15. Objeto Restaurante

Al cargar la pantalla de selección de tarjetas, realizaremos la llamada a Places API, que nos devolverá todos los restaurantes comprendidos en el radio seleccionado en la pantalla del mapa. Cuando se complete la llamada, estaremos trabajando con una lista de Restaurantes, donde cada uno contendrá imágenes del sitio y de sus platos. A continuación, desplegamos todas las imágenes en una lista de mapas, cada mapa será un par con la url de la imagen y el nombre del restaurante al que pertenece. Será esta lista de mapas con la que trabajaremos ahora en lugar de la lista 'originalCards' de la primera versión de la pantalla de selección.

```
void initState() {
  super.initState();
  final location = Provider.of<Location>(context, listen: false);
  location.getNearbyRestaurants().then((restaurants) {
    setState(() {
      originalRestaurants = restaurants;
      for (var restaurant in restaurants) {
        print("${restaurant.name}: ${restaurant.photos.length}");
        for (var photoReference in restaurant.photos) {
          final imageUrl = restaurant.getPhotoUrl(
              'AIzaSyBxTuZzg44MYeFcoMZnNsZzdWiNu2YsoWY', photoReference);
          if (!addedImages.contains(imageUrl)) {
            imagesWithRestaurant.add({
              'imageUrl': imageUrl,
              'restaurantName': restaurant.name,
            });
            addedImages.add(imageUrl); // Agregar la imagen al conjunto
          }
        }
      }
      imagesWithRestaurant.shuffle();
      dismissibleKeys =
          List.generate(imagesWithRestaurant.length, (index) => GlobalKey());
    });
  });
}
```

Figura 16. Nueva inicialización de la pantalla de selección de tarjetas.

La otra diferencia con la primera versión de la pantalla es que cambiamos el texto temporal "Contenido de la tarjeta" por la imagen que hemos obtenido. El resultado de esta implementación se puede ver en la Figura 17.

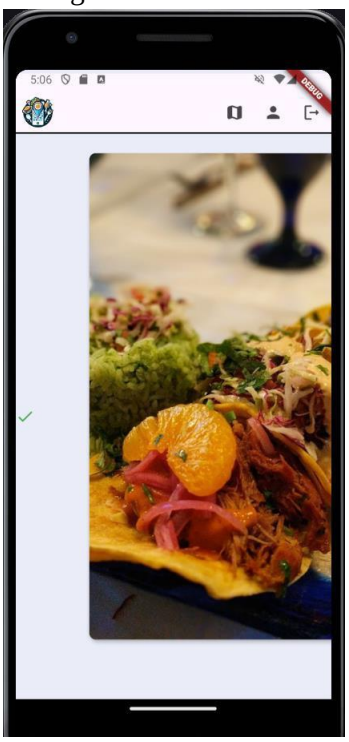


Figura 17. Pantalla de selección con la Places API implementada.

Implementación del algoritmo "match-making"

En esta sección se explicará cómo se ha llegado a la forma de realizar los "matches" y cuál ha sido la línea de pensamiento.

Trabajar con la API de Places nos da la ventaja de conocer las etiquetas que tienen asociadas los restaurantes. A cada etiqueta le podemos asociar un valor, un peso, que dependan de las preferencias del usuario. Estos valores podemos hacer que evolucionen con el uso de la aplicación y las elecciones del usuario. Cada vez que se realice un "match", las etiquetas que han sido seleccionadas a lo largo del proceso retroalimentarán estos valores, haciendo que se vaya adaptando y condicionando las nuevas recomendaciones. Esta idea se puede ver implementada en la Figura 18.

Figura 18. Ejemplo de los pesos para un usuario registrado.

Sin embargo, no sería correcto basar la recomendación únicamente en las elecciones pasadas; también debemos dar cabida a que el usuario experimente. Además, tenemos en cuenta la opinión de otros usuarios, moderada con las preferencias propias del usuario.

Para cumplir estas necesidades se propone la fórmula de la Figura 19:

$$P = \frac{\sum_i \left(\frac{n_i}{1-u_i^2}\right)}{tT} + \frac{N}{T} + \frac{R(r+1)}{10}$$

Figura 19. Fórmula para calcular la puntuación de un restaurante.

Variables y Parámetros:

- **P:** Puntuación total que determina la similitud entre el usuario y el restaurante.
- **r:** Puntuación del restaurante que proviene del modelo de recomendación.
- **N:** Número total de "likes" que ha recibido el restaurante.
- **n:** Número de "likes" que ha recibido una etiqueta específica.
- **u:** Valor de la etiqueta en el modelo de usuario.
- **R:** Calificación global del restaurante.
- **t:** Número total de etiquetas asociadas al restaurante.
- **T:** Número total de likes realizados hasta llegar al match.

Desglose de la Fórmula

La fórmula está compuesta por tres sumandos, cada cual tendrá en cuenta distintos parámetros.

- **Sumatorio ponderado del usuario:** las etiquetas que tengan un valor de *u* más alto darán más peso a las elecciones. Como *u* siempre es menor que 0, al elevarlo a 2, será menor. Por tanto, el valor de *n* será mayor. Esta parte la dividimos entre el producto del número de etiquetas del restaurante y los likes totales del usuario.
- **Ratio de likes sobre likes totales:** Con esta fracción tenemos en cuenta la popularidad del restaurante en comparación con el resto de las opciones.

- **Rating del restaurante ajustado por la puntuación del modelo:** Ponderamos el rating general del restaurante con las preferencias del usuario y ajustamos para que pueda dar como máximo 1.

(Los registros de las pruebas de la evolución del algoritmo y de las elecciones se pueden encontrar en el Anexo)

Sin embargo, nos encontramos con el problema de cuándo evaluar las puntuaciones, ¿qué valor es el mínimo para considerarlo un "match"? La solución fue limitar el número de likes.

Al usuario le daremos 3 opciones distintas para encontrar una recomendación: rápida, estándar y precisa. En función de la elegida, se asignarán puntuaciones a los restaurantes después de un número distinto de likes: 5, 10 y 15, respectivamente. Cuantos más likes del usuario, más seguro podemos estar de la recomendación ya que tenemos más datos con los que trabajar.

2.4. . Resultados y validación

En esta sección se presentan los resultados obtenidos del desarrollo y la validación de ZampApp, junto con un análisis detallado de su funcionamiento y rendimiento. Se abordarán aspectos clave como el tiempo de ejecución, el rendimiento de la aplicación en diferentes dispositivos y los problemas detectados durante el proceso de desarrollo y prueba. Además, se incluirán los resultados de las pruebas realizadas para asegurar la efectividad y fiabilidad de la aplicación.

Funcionamiento de la Aplicación

Atendiendo al funcionamiento de la aplicación, a continuación, se detallan los aspectos más relevantes de su funcionamiento:

1. **Tiempo de Ejecución**: La aplicación tarda varios segundos en iniciar. Esto es debido a diversos factores entre los que se encuentran: la inicialización de todos los widgets de Flutter, los servicios de Firebase y los de geolocalización que se lanzan al iniciar.
2. **Rendimiento**: En general la aplicación responde bien y las cargas de los restaurantes son relativamente rápidas con una buena conexión a Internet. Sin embargo, la pantalla del mapa es más lenta que las llamadas para obtener los restaurantes.

Resultados de las Pruebas y Simulaciones

Para validar el funcionamiento de la aplicación, se realizaron extensas pruebas en diferentes escenarios y condiciones. A continuación, se describen los resultados de las pruebas:

1. **Pruebas en Diferentes Dispositivos**
   - **Condiciones de Prueba**: Se probó la aplicación en un dispositivo físico con Android 10 y en un emulador con diferentes versiones (8.0 y superiores) para evaluar su rendimiento.
   - **Resultados**: ZampApp funcionó de manera consistente en ambos entornos, con ligeras variaciones en el tiempo de carga inicial y el rendimiento de la animación. Estas variaciones se mantuvieron dentro de límites aceptables, sin afectar significativamente la experiencia del usuario.

2. **Pruebas de Uso Real**
   - **Condiciones de Prueba**: Pruebas de la aplicación para distintos usuarios en distintas localizaciones.
   - **Resultados**: Los restaurantes mostrados cambian en función de la ubicación y el rango seleccionado. Además, según los usuarios y los likes dados, las recomendaciones cambian. Para un mismo usuario, en la misma localización y distintas elecciones, las recomendaciones cambian acorde a la fórmula utilizada.
3. **Pruebas unitarias**
   - **Condiciones de Prueba**: Pruebas unitarias para los métodos: calcRank y normalizeUserModel

```
test('normalizeUserModel deberia normalizar el user model de manera correcta', () {
  final user = ZampUser(
    uuid: 'some_uuid',
    userModel: {
      'category1': 0.2,
      'category2': 0.3,
      'category3': 0.5,
    },
  );

  final newValues = {
    'category1': 0.1,
    'category2': 0.2,
    'category3': 0.3,
  };

  final expectedUserModel = {
    'category1': (0.2 + 0.1) / (0.2 + 0.3 + 0.5 + 0.1 + 0.2 + 0.3),
    'category2': (0.3 + 0.2) / (0.2 + 0.3 + 0.5 + 0.1 + 0.2 + 0.3),
    'category3': (0.5 + 0.3) / (0.2 + 0.3 + 0.5 + 0.1 + 0.2 + 0.3),
  };

  user.normalizeUserModel(newValues);

  expect(user.userModel, expectedUserModel);
});
```

Figura 20. Test del método 'normalizeUserModel'

```dart
test('Calcular el rango del restaurante', () {
  final restaurante = Restaurante(
    name: 'Restaurante Prueba',
    address: 'Dirección de Prueba',
    isOpen: true,
    website: 'https://www.restauranteprueba.com',
    phoneNumber: '123456789',
    photos: ['photo1', 'photo2'],
    tags: ['tag1', 'tag2', 'tag3'],
    rating: 4.5,
    ratingCount: 100,
  );

  final userModel = {
    'tag1': 0.3,
    'tag2': 0.5,
    'tag3': 0.2,
  };

  // Ejecución
  final rank = restaurante.calcRank(userModel);

  // Verificación
  expect(rank, equals(1.0));
});
```

Figura 21. Test del método 'calcRank'

- **Resultados**:

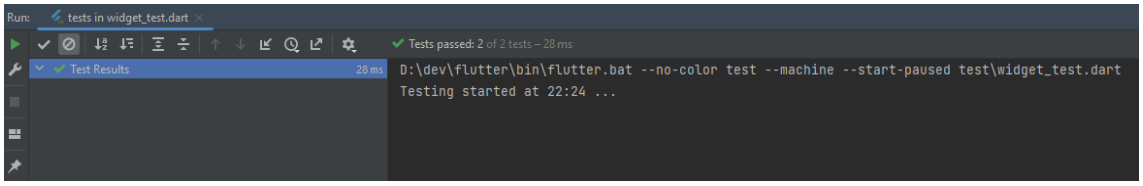


Figura 22. Resultados de las pruebas unitarias.

3. CONCLUSIONES

En este Trabajo de Fin de Ciclo (TFC) se ha desarrollado ZampApp, una aplicación móvil diseñada para simplificar la decisión de dónde comer mediante una interfaz interactiva que permite a los usuarios seleccionar o descartar restaurantes deslizando tarjetas con imágenes de los lugares y/o los platos que ofrecen. Este enfoque innovador no solo facilita la toma de decisiones, sino que también proporciona una experiencia de usuario atractiva y personalizada.

Durante el desarrollo de ZampApp, se han logrado varios hitos importantes. Se ha conseguido implementar una interfaz de usuario intuitiva y visualmente atractiva haciendo uso de Flutter. Además, se ha integrado exitosamente la API de Google Maps, lo que permite extraer información precisa y actualizada de los restaurantes cercanos al usuario.

Sin embargo, durante el desarrollo del proyecto se identificaron varios desafíos técnicos al trabajar con Flutter. Uno de los problemas fue que la inicialización de la aplicación, Flutter resultaba considerablemente más lenta en comparación con otros IDEs convencionales, ya que este utiliza su propio motor de renderizado y un entorno de ejecución para el código Dart, por lo que cuando la aplicación se inicia, debe cargar e inicializar estos componentes antes de que el código pueda ejecutarse. Además, las llamadas a la API de Google Maps experimentaban un retraso significativo, lo que incrementaba el tiempo de carga de los restaurantes. Para mitigar este problema, se implementó tres botones los cuales, dependiendo de la precisión que quisiera el usuario, limitaban el número de “swipes” y por consiguiente realizando un menor número de llamadas a la API, acelerando notablemente el tiempo de carga de los restaurantes.

A pesar de estos obstáculos, se han cumplido los objetivos establecidos inicialmente. La aplicación es funcional y cumple con su propósito de facilitar la elección de restaurantes de manera interactiva.

El resultado obtenido es fiable, aunque se sugiere la realización de pruebas adicionales en diferentes entornos y con una mayor base de usuarios para identificar posibles mejoras y optimizaciones.

En resumen, ZampApp representa una solución efectiva y atractiva para resolver el dilema de dónde comer, aprovechando tecnologías avanzadas para ofrecer una experiencia de usuario intuitiva y diferente a la manera actual en la que la gran mayoría de personas deciden dónde y qué comer. Con algunas pruebas y mejoras adicionales, la aplicación tiene el potencial de convertirse en una herramienta indispensable para los usuarios en sus decisiones gastronómicas.

3.1. . Aportaciones

La realización del proyecto ZampApp ha introducido varios aspectos novedosos que merecen ser destacados:

- **Interfaz de Usuario Interactiva y Atractiva:** La implementación de una interfaz de usuario basada en tarjetas deslizables es una característica innovadora que mejora significativamente la experiencia del usuario. Este no solo simplifica el proceso de

selección de restaurantes, sino que también añade un elemento visualmente atractivo, haciendo que la interacción con la aplicación sea más agradable, efectiva y cómoda.

- **Personalización de Recomendaciones**: ZampApp ofrece recomendaciones de restaurantes personalizadas basadas en las preferencias del usuario. A través del análisis de las elecciones y descartes de los usuarios, la aplicación es capaz de ajustar sus sugerencias para adaptarse mejor a los gustos individuales. Esta personalización incrementa la relevancia de las recomendaciones y mejora la satisfacción del usuario. Atendiendo al impacto.
- **Integración de la API de Google Maps**: La integración eficiente de la API de Google Maps permite proporcionar información precisa y actualizada sobre los restaurantes cercanos al usuario. Esta característica facilita la localización y selección de restaurantes, ofreciendo una ventaja competitiva frente a otras aplicaciones que no disponen de información geolocalizada tan detallada y en tiempo real.

En cuanto al impacto, la interfaz interactiva y la personalización tienen como objetivo aumentar el compromiso y la satisfacción del usuario, lo que puede traducirse en una mayor retención y uso continuo de la aplicación.

No obstante, hay que resaltar el **Valor Añadido para la Industria Gastronómica** , la integración de la API de Google Maps puede influir en cómo los restaurantes se promocionan en Google Maps, es decir, renovando y actualizando periódicamente imágenes, información, etc. con el fin de atraer nuevos clientes.

3.2. . Trabajo futuro

A pesar de los avances y logros alcanzados con el desarrollo de ZampApp, existen diversas áreas en las que el proyecto puede expandirse y mejorarse. A continuación, se presentan algunas recomendaciones y propuestas para el trabajo futuro:

- **Colaboraciones directas con Restaurantes:** Establecer asociaciones con restaurantes locales para obtener información exclusiva sobre menús, promociones y eventos especiales.
- **Análisis de Opiniones y Valoraciones**: Incorporar análisis de opiniones y valoraciones de otros usuarios para afinar aún más las sugerencias de restaurantes.
- **Reservas y Pedidos en Línea:** Integrar opciones para realizar reservas directamente desde la aplicación y, potencialmente, realizar pedidos en línea para restaurantes que ofrezcan servicios de entrega o recogida.
- **Soporte para diferentes idiomas:** Ampliar la aplicación para que soporte múltiples idiomas, lo que permitirá su uso en diferentes regiones del mundo.

4. BIBLIOGRAFÍA Y WEBGRAFÍA

2.1 - Material

Flutter: guía oficial de inicio rápido. Disponible en: https://flutter.dev/docs/get-started/install
Flutter Cookbook: Recetas útiles para el desarrollo con Flutter. Disponible en: https://f lutter.dev/docs/cookbook
Documentación oficial de Flutter. Disponible en: https://flutter.dev/docs

2.1 - Dart

Dart: guía oficial de inicio rápido. Disponible en: https://dart.dev/guides/get-started
Documentación oficial de Dart. Disponible en: https://dart.dev/guides

2.1 - FireBase

Documentación oficial de Firebase. Disponible en: https://firebase.google.com/docs
Cómo empezar con Firebase Firestore. Disponible en: https://f irebase.google.com/docs/f irestore/quickstart
Cómo empezar con Firebase Authentication. Disponible en: https://f irebase.google.com/docs/auth

2.1 -Google Maps API

Documentación oficial de Google Maps Platform. Disponible en: https://developers.google.com/maps/documentation
Cómo empezar con Google Maps API en Flutter. Disponible en: https://pub.dev/packages/google_maps_f lutter

5. ANEXOS

5.1. . Registros de la evolución del algoritmo y las elecciones

A continuación, se presentan los registros del desarrollo del algoritmo para un modelo inicial de un usuario de prueba. La prueba se realizó para 10 iteraciones en las que el usuario elegía los mismos restaurantes. Se puede observar cómo los valores del modelo cambian, así como el orden de las recomendaciones a lo largo de las iteraciones.

```
Iteración nº1
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante
9': 4}
Modelo de usuario: {'a': 0.5714285714285714, 'b': 0.2857142857142857, 'e': 0.14285714285714285}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.499074074074074, p2=0.041666666666666664,
p3=0.7800000000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.34027777777777773, p2=0.0, p3=0.8057142857142857
Restaurante 3: tags=['e', 'b', 'c'], p1=0.38564814814814813, p2=0.0, p3=0.5
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.36749999999999994, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.34027777777777773, p2=0.0, p3=0.6757142857142857
Restaurante 6: tags=['b', 'd'], p1=0.40833333333333327, p2=0.375, p3=0.5142857142857142
Restaurante 7: tags=['c', 'e'], p1=0.17013888888888887, p2=0.0, p3=0.4342857142857143
Restaurante 8: tags=['a', 'b', 'e'], p1=0.6124999999999999, p2=0.16666666666666666, p3=0.9
Restaurante 9: tags=['b', 'd', 'e'], p1=0.38564814814814813, p2=0.16666666666666666,
p3=0.5857142857142857
Restaurante 10: tags=['a', 'c', 'd'], p1=0.22685185185185183, p2=0.25, p3=0.6128571428571429
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 1.0): 1.6791666666666667
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.8571428571428571): 1.3207407407407408
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.2857142857142857): 1.2976190476190474
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.7142857142857142): 1.1459920634920635
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.42857142857142855): 1.1380291005291006
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5714285714285714): 1.0897089947089946
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.5714285714285714): 1.0159920634920634
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.0074999999999998
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.42857142857142855): 0.8856481481481482
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.14285714285714285): 0.6044246031746031
Iteración nº2
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante
9': 4}
Modelo de usuario: {'a': 0.373015873015873, 'b': 0.2857142857142857, 'e': 0.1349206349206349,
'c': 0.05555555555555555, 'd': 0.15079365079365079}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5472167232247275, p2=0.041666666666666664,
p3=0.7200000000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5606715178150452, p2=0.0, p3=0.7796031746031746
Restaurante 3: tags=['e', 'b', 'c'], p1=0.4829166744737512, p2=0.0, p3=0.5166666666666667
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5582501759109054, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.4124917515037578, p2=0.0, p3=0.6142857142857143
Restaurante 6: tags=['b', 'd'], p1=0.8133768396175743, p2=0.375, p3=0.5746031746031747
Restaurante 7: tags=['c', 'e'], p1=0.31604167837729347, p2=0.0, p3=0.4523809523809524
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5628647358477734, p2=0.16666666666666666,
p3=0.807142857142857
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6554224588490036, p2=0.16666666666666666,
p3=0.6442857142857142
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5450235051919993, p2=0.25, p3=0.6159523809523809
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.4365079365079365): 1.7629800142207492
```

```
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.7936507936507936): 1.536674259657297
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.5714285714285714): 1.4663748398013845
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5793650793650793): 1.4109758861443802
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6587301587301587): 1.3402746924182198
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.7142857142857143): 1.308883389891394
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1982501759109054
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.4285714285714286): 1.0267774657894722
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.47619047619047616): 0.9995833411404179
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.19047619047619047): 0.7684226307582458
Iteración nº3
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante
9': 4}
Modelo de usuario: {'a': 0.27380952380952384, 'b': 0.2857142857142857, 'e':
0.13095238095238093, 'c': 0.08333333333333333, 'd': 0.22619047619047616}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5352844243069461, p2=0.041666666666666664,
p3=0.6900000000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5563278503457874, p2=0.0, p3=0.766547619047619
Restaurante 3: tags=['e', 'b', 'c'], p1=0.48317406778113337, p2=0.0, p3=0.5249999999999999
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5558713022820644, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.3945933031270859, p2=0.0, p3=0.5835714285714286
Restaurante 6: tags=['b', 'd'], p1=0.8255103310928553, p2=0.375, p3=0.6047619047619048
Restaurante 7: tags=['c', 'e'], p1=0.3164277683383667, p2=0.0, p3=0.4614285714285714
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5504320740616615, p2=0.16666666666666666,
p3=0.7607142857142857
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6633899683853836, p2=0.16666666666666666,
p3=0.6735714285714285
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5411802005910719, p2=0.25, p3=0.6175
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5119047619047619): 1.8052722358547602
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.6428571428571428): 1.5036280636234787
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.6904761904761905): 1.4778130264426137
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5833333333333334): 1.4086802005910 72
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6309523809523809): 1.3228754693934064
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.6428571428571429): 1.2669510909736128
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1958713022820644
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.49999999999999994): 1.0081740677811333
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.35714285714285715): 0.9781647316985145
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.21428571428571425): 0.7778563397669381
Iteración nº4
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante
9': 4}
Modelo de usuario: {'a': 0.22420634920634924, 'b': 0.28571428571428575, 'e':
0.12896825396825398, 'c': 0.09722222222222222, 'd': 0.2638888888888889}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5312363500860762, p2=0.041666666666666664, p3=0.675
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5574955239883985, p2=0.0, p3=0.7600198412698413
Restaurante 3: tags=['e', 'b', 'c'], p1=0.483362633561311, p2=0.0, p3=0.5291666666666666
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5567206185150775, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.38852119179578093, p2=0.0, p3=0.5682142857142857
Restaurante 6: tags=['b', 'd'], p1=0.8337946644550418, p2=0.375, p3=0.6198412698412699
Restaurante 7: tags=['c', 'e'], p1=0.31671061700863323, p2=0.0, p3=0.465952380952381
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5460768587961484, p2=0.16666666666666666, p3=0.7375
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6688535696612751, p2=0.16666666666666666,
p3=0.6882142857142857
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5426550152783264, p2=0.25, p3=0.6182738095238096
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5496031746031746): 1.8286359342963117
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.6785714285714286): 1.5237345220422274
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.638888888888889): 1.450243525462815
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5853174603174603): 1.410928824802136
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6170634920634921): 1.3175153652582399
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.6071428571428572): 1.2479301675 7427
```

```
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1967206185150774
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.511904761904762): 1.0125293002279776
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.32142857142857145): 0.9567354775100666
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.22619047619047622): 0.7826629979610142
Iteración nº5
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante 9': 4}
Modelo de usuario: {'a': 0.19940476190476192, 'b': 0.2857142857142857, 'e': 0.12797619047619047, 'c': 0.10416666666666666, 'd': 0.28273809523809523}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5296150622392201, p2=0.041666666666666664, p3=0.6675000000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5588825977235243, p2=0.0, p3=0.7567559523809525
Restaurante 3: tags=['e', 'b', 'c'], p1=0.4834721239464517, p2=0.0, p3=0.5312499999999999
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5576361258419542, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.3860892600254968, p2=0.0, p3=0.5605357142857142
Restaurante 6: tags=['b', 'd'], p1=0.8385592866793106, p2=0.375, p3=0.6273809523809524
Restaurante 7: tags=['c', 'e'], p1=0.3168748525863443, p2=0.0, p3=0.46821428571428575
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5442875177150951, p2=0.16666666666666666, p3=0.7258928571428572
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6720007030529259, p2=0.16666666666666666, p3=0.6955357142857143
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5442101422476494, p2=0.25, p3=0.6186607142857142
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5684523809523809): 1.840940239060263
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.6964285714285714): 1.534203084005307
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.6130952380952381): 1.436847041524619
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5863095238095238): 1.4128708565333636
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6101190476190477): 1.315638550104477
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.5892857142857143): 1.2387817289058867
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1976361258419543
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.5178571428571428): 1.0147221239464517
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.3035714285714286): 0.9466249743112111
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.23214285714285712): 0.78508913830063
Iteración nº6
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante 9': 4}
Modelo de usuario: {'a': 0.18700396825396828, 'b': 0.28571428571428575, 'e': 0.12748015873015875, 'c': 0.1076388888888889, 'd': 0.29216269841269843}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5288981761759443, p2=0.041666666666666664, p3=0.6637500000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5597770966995445, p2=0.0, p3=0.755124007936508
Restaurante 3: tags=['e', 'b', 'c'], p1=0.48353070605499876, p2=0.0, p3=0.5322916666666667
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5582167046570378, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.38501393093058306, p2=0.0, p3=0.5566964285714285
Restaurante 6: tags=['b', 'd'], p1=0.8411078882227904, p2=0.375, p3=0.6311507936507937
Restaurante 7: tags=['c', 'e'], p1=0.31696272574916495, p2=0.0, p3=0.4693452380952381
Restaurante 8: tags=['a', 'b', 'e'], p1=0.543482948995462, p2=0.16666666666666666, p3=0.7200892857142858
Restaurante 9: tags=['b', 'd', 'e'], p1=0.673685220474674, p2=0.16666666666666666, p3=0.6991964285714285
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5451923238800267, p2=0.25, p3=0.6188541666666667
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5778769841269842): 1.8472586818735843
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.7053571428571429): 1.5395483157127692
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.6001984126984128): 1.4302389013764145
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5868055555555556): 1.4140464905466934
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6066468253968255): 1.3149011046360526
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.5803571428571429): 1.234314842842611
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1982167046570378
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.5208333333333334): 1.0158223727216655
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.2946428571428572): 0.9417103595020115
```

```
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.23511904761904764): 0.786307963844403
Iteración nº7
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante 9': 4}
Modelo de usuario: {'a': 0.18080357142857145, 'b': 0.2857142857142857, 'e': 0.12723214285714285, 'c': 0.109375, 'd': 0.296875}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5285624387108635, p2=0.041666666666666664, p3=0.6618750000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5602748447268471, p2=0.0, p3=0.7543080357142858
Restaurante 3: tags=['e', 'b', 'c'], p1=0.4835609610410698, p2=0.0, p3=0.5328125
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5585378579933441, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.3845103247329618, p2=0.0, p3=0.5547767857142857
Restaurante 6: tags=['b', 'd'], p1=0.8424252565818829, p2=0.375, p3=0.6330357142857143
Restaurante 7: tags=['c', 'e'], p1=0.3170081082282714, p2=0.0, p3=0.4699107142857143
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5431024514500361, p2=0.16666666666666666, p3=0.7171875
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6745562135002655, p2=0.16666666666666666, p3=0.7010267857142857
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5457348319876744, p2=0.25, p3=0.6189508928571429
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5825892857142857): 1.850460970867597
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.7098214285714286): 1.5422496658812177
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.59375): 1.4269566181167028
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5870535714285714): 1.4146857248448172
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6049107142857143): 1.314582880441133
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.5758928571428572): 1.2321041053775303
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1985378579933441
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.5223214285714286): 1.01637346104107
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.29017857142857145): 0.9392871104472476
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.23660714285714285): 0.7869188225139857
Iteración nº8
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante 9': 4}
Modelo de usuario: {'a': 0.17770337301587302, 'b': 0.2857142857142857, 'e': 0.1271081349206349, 'c': 0.11024305555555555, 'd': 0.2992311507936508}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5284001621050395, p2=0.041666666666666664, p3=0.6609375000000001
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5605363903975403, p2=0.0, p3=0.7539000496031746
Restaurante 3: tags=['e', 'b', 'c'], p1=0.48357633012537854, p2=0.0, p3=0.5330729166666667
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5587061792234908, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.384266909824226, p2=0.0, p3=0.5538169642857143
Restaurante 6: tags=['b', 'd'], p1=0.8430949055088499, p2=0.375, p3=0.6339781746031746
Restaurante 7: tags=['c', 'e'], p1=0.3170311618547345, p2=0.0, p3=0.4701934523809524
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5429175645027515, p2=0.16666666666666666, p3=0.7157366071428571
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6749990254896674, p2=0.16666666666666666, p3=0.7019419642857142
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5460189879998284, p2=0.25, p3=0.618999255952381
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5849454365079365): 1.8520730801120244
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.7120535714285714): 1.5436076564420482
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.5905257936507936): 1.4253208383122753
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5871775793650794): 1.4150182439522094
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6040426587301587): 1.314436440000715
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.5736607142857143): 1.2310043287717063
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1987061792234908
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.5230654761904762): 1.0166492467920452
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.2879464285714286): 0.9380838741099402
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.23735119047619047): 0.7872246142356869
Iteración nº9
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
```

```
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante 9': 4}
Modelo de usuario: {'a': 0.17615327380952384, 'b': 0.28571428571428575, 'e': 0.12704613095238096, 'c': 0.11067708333333334, 'd': 0.3004092261904762}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5283204117462452, p2=0.041666666666666664, p3=0.66046875
Restaurante 2: tags=['a', 'd', 'e'], p1=0.5606703379486703, p2=0.0, p3=0.7536960565476191
Restaurante 3: tags=['e', 'b', 'c'], p1=0.48358407514268165, p2=0.0, p3=0.533203125
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5587922801065589, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.3841472842860345, p2=0.0, p3=0.5533370535714286
Restaurante 6: tags=['b', 'd'], p1=0.8434324966097319, p2=0.375, p3=0.6344494047619048
Restaurante 7: tags=['c', 'e'], p1=0.31704277938068925, p2=0.0, p3=0.47033482142857147
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5428264513199602, p2=0.16666666666666666, p3=0.7150111607142858
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6752222773202418, p2=0.16666666666666666, p3=0.7023995535714285
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5461642983749554, p2=0.25, p3=0.6190234375000001
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.586123511904762): 1.8528819013716369
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.7131696428571429): 1.5442884975583369
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.5889136904761906): 1.4245042787009128
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5872395833333334): 1.4151877358749556
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.603608630952381): 1.3143663944962893
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.5725446428571429): 1.230455828412912
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.198792280106559
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.5234375): 1.0167872001426816
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.2868303571428572): 0.9374843378574631
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.2377232142857143): 0.7873776008092608
Iteración nº10
Etiquetas seleccionadas: {'a': 11, 'b': 18, 'c': 7, 'd': 19, 'e': 8}
{'Restaurante 1': 1, 'Restaurante 10': 6, 'Restaurante 8': 4, 'Restaurante 6': 9, 'Restaurante 9': 4}
Modelo de usuario: {'a': 0.17537822420634921, 'b': 0.2857142857142857, 'e': 0.12701512896825395, 'c': 0.11089409722222222, 'd': 0.3009982638888889}
Restaurante 1: tags=['a', 'b', 'c'], p1=0.5282808823191555, p2=0.041666666666666664, p3=0.660234375
Restaurante 2: tags=['a', 'd', 'e'], p1=0.560738106311209, p2=0.0, p3=0.7535940600198413
Restaurante 3: tags=['e', 'b', 'c'], p1=0.48358796277986776, p2=0.0, p3=0.5332682291666666
Restaurante 4: tags=['a', 'b', 'c', 'd', 'e'], p1=0.5588358161657795, p2=0.0, p3=0.64
Restaurante 5: tags=['a', 'c'], p1=0.38408799014539996, p2=0.0, p3=0.5530970982142857
Restaurante 6: tags=['b', 'd'], p1=0.8436019870468826, p2=0.375, p3=0.6346850198412699
Restaurante 7: tags=['c', 'e'], p1=0.31704861083646835, p2=0.0, p3=0.470405505952381
Restaurante 8: tags=['a', 'b', 'e'], p1=0.5427812260577315, p2=0.16666666666666666, p3=0.7146484375
Restaurante 9: tags=['b', 'd', 'e'], p1=0.6753343668460325, p2=0.16666666666666666, p3=0.7026283482142857
Restaurante 10: tags=['a', 'c', 'd'], p1=0.5462377625726329, p2=0.25, p3=0.6190355282738096
Restaurante 6(Likes: 9, Rating: 4.0, Rank: 0.5867125496031746): 1.8532870068881524
Restaurante 9(Likes: 4, Rating: 4.1, Rank: 0.7137276785714286): 1.5446293817269847
Restaurante 8(Likes: 4, Rating: 4.5, Rank: 0.5881076388888888): 1.424096330224398
Restaurante 10(Likes: 6, Rating: 3.9, Rank: 0.5872705853174603): 1.4152732908464425
Restaurante 2(Likes: 0, Rating: 4.7, Rank: 0.6033916170634921): 1.3143321663310503
Restaurante 1(Likes: 1, Rating: 4.2, Rank: 0.5719866071428571): 1.230181923985822
Restaurante 4(Likes: 0, Rating: 3.2, Rank: 1.0): 1.1988358161657795
Restaurante 3(Likes: 0, Rating: 3.5, Rank: 0.5236235119047619): 1.0168561919465344
Restaurante 5(Likes: 0, Rating: 4.3, Rank: 0.28627232142857145): 0.9371850883596856
Restaurante 7(Likes: 0, Rating: 3.8, Rank: 0.23790922619047616): 0.7874541167888494
```