Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

Universidad de Jaén

Escuela Politécnica Superior de Jaén

PROTOTIPO DE APLICACIÓN WEB PARA UNA RED SOCIAL DE NEGOCIOS

Alumno: Joaquín Cañete Godino

Tutor: Prof. D. José Ramón Balsas Almagro

Dpto: Departamento de Informática

Septiembre,2021

Contenidos

0.2 Índice figuras

1 Introducción

1.1 Motivación

En este trabajo se planificará, diseñará e implementará una aplicación web de una red social orientada a negocios. Dado que el propósito de la propuesta original del TFG era una aplicación web, mi intención era desarrollarla orientándola a un red social, en un contexto en el que estas son cada vez son más relevantes. Además, este proceso me permitiría poder ampliar mis conocimientos de diversas tecnologías y metodologías de trabajo que se han empleado durante el grado. A su vez, mientras las redes sociales se van infiltrando en el día a día de pequeñas y grandes empresas, muchas empresas pequeñas y medianas no pueden competir en las mismas condiciones que las otras a la hora de hacerse ver, ya que las grandes redes sociales promocionan a los gigantes muy por encima de los pequeños negocios, haciéndolos casi invisibles. Por lo tanto, con el fin de intentar plantear una alternativa diferente junto con los otros motivos expuestos antes, decidí enfocar mi trabajo a una red social para estos pequeños negocios.

1.2 Objetivos

Para la evaluación de la correcta finalización de este TFG se muestran los tres objetivos principales en su propuesta que deben ser cumplidos. Estos objetivos son los siguientes:

- Realizar un estudio de necesidades y soluciones existentes en el contexto seleccionado.
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas, principios de diseño y metodologías de desarrollo web.
- Seleccionar un entorno de desarrollo web e implementar un prototipo de aplicación web que satisfaga las necesidades que se hayan determinado.

1.3 Metodología

La metodología que se va a usar en este trabajo es una metodología de desarrollo iterativa basada en Scrum. Scrum es una metodología de trabajo ágil en la que el trabajo se organiza en iteraciones cortas, con una duración de entre una y cuatro semanas, llamadas Sprints. Para cada uno de estos Sprints se seleccionan una serie de historias de usuario que han de ser priorizadas y puntuadas, es decir, se cuantifica el esfuerzo que cuesta desarrollarlas y cómo de prioritarias son para el producto final. Al final de cada Sprint se obtiene un entregable funcional lo que permite ir construyendo progresivamente las aplicaciones, teniendo siempre un producto. Dentro de los equipos Scrum existen varios roles, que en el caso de este TFG son desempeñados todos por una única persona, estos roles son:

- El Product Owner, es el responsable desde el punto de vista del negocio
- El Scrum Master, es el responsable de organizar al equipo de desarrollo para alcanzar los objetivos acordados
- El equipo, son los responsables de desarrollar el producto final.

1.4 Organización del documento

Este documento se encuentra estructurado en función del clásico orden de tareas en cualquier proceso de desarrollo. Primero, en el apartado 2 Análisis se realizará un análisis del contexto de viabilidad de la aplicación, así como una estimación de las historias de usuario que hay que desarrollar, su planificación y su coste de desarrollo. En el apartado 3 Diseño se presentarán los diagramas y las justificaciones de las decisiones de diseño del sistema planteado así como el diseño de las pruebas que se realizarán para la validación de las historias de usuario. En el apartado 4 Implementación y despliegue en la nube se comentarán detalles de la implementación que son relevantes para el mantenimiento y ampliación del sistema, así como los pasos necesarios para desplegar tanto el back-end como el front-end de la aplicación. A continuación, en el apartado 5 se explicarán las pruebas realizadas, así como resultados relevantes de las mismas. Por último, se incluirá un apartado de conclusiones donde se reflexionará sobre el trabajo realizado. Tras esto se incluirán una serie de apéndices adicionales que detallarán un manual de usuario, un manual de instalación del sistema, un listado con los documentos adicionales que se entregarán junto con esta memoria así como una serie de instrucciones para probar el sistema desarrollado.

2 Análisis

2.1 Análisis preliminar

En el contexto actual en el que las redes sociales cobran más importancia día a día, tanto entre las relaciones entre las personas como en las formas en las que las empresas se relacionan con sus clientes, vemos como hay multitud de redes sociales como Twitter, Instagram o Facebook que pese a no estar originalmente orientadas a negocios son utilizadas por estos para su promoción. En este trabajo se plantea una aplicación web que intenta aunar diferentes elementos de redes sociales con la comodidad y potencia para localizar negocios a tu alrededor del motor de búsqueda de Google Maps.

Mientras que otras redes sociales tratan más sobre cómo las personas se relacionan entre sí, y otras aplicaciones como Google Maps simplemente nos sirven para localizar negocios, la solución de este trabajo intenta alcanzar un camino intermedio, incorporando elementos como los comentarios en formato “tweet” para los negocios Esto permitirá seguir a negocios para obtener sus últimas novedades, seguimiento de eventos e incluso ofertas que puedan realizar para sus seguidores además de la facilidad de poder localizar negocios que te interesen en tu zona.

2.1.1 Soluciones similares

Dado el enfoque de red social propuesto para la solución de este TFG, conviene observar otras soluciones similares que provean servicios parecidos a los que se plantean, así como sus puntos débiles y fuertes que se tendrán en cuenta para integrarlos, mejorarlos o evitarlos en la solución final. Algunas de las soluciones más interesantes que se desglosarán con más detalle a continuación serían las siguientes, para las que mostrarán algunas imágenes sobre las mismas para facilitar su visualización: *TripAdvisor, Around Me y Yelp*

- **TripAdvisor**: Es una aplicación de búsqueda de negocios y página web, con el propósito principal de facilitar el hacer reservas en los negocios registrados. Su modelo de negocio está más orientado a restaurantes, hoteles, spas y sitios turísticos respecto al enfoque más genérico que se plantea en este TFG

    - **Fortalezas**: Años de experiencia en el sector le han permitido desarrollar funcionalidades que dan gran valor a la aplicación, la posibilidad de hacer reservas directamente en los negocios y sus mapas personalizados le dan un gran valor agregado.
    - **Debilidades**:El enfoque tan específico de la aplicación hace que su uso, fuera del contexto de hacer reservas para negocios que lo permitan, carezca de sentido y no permite

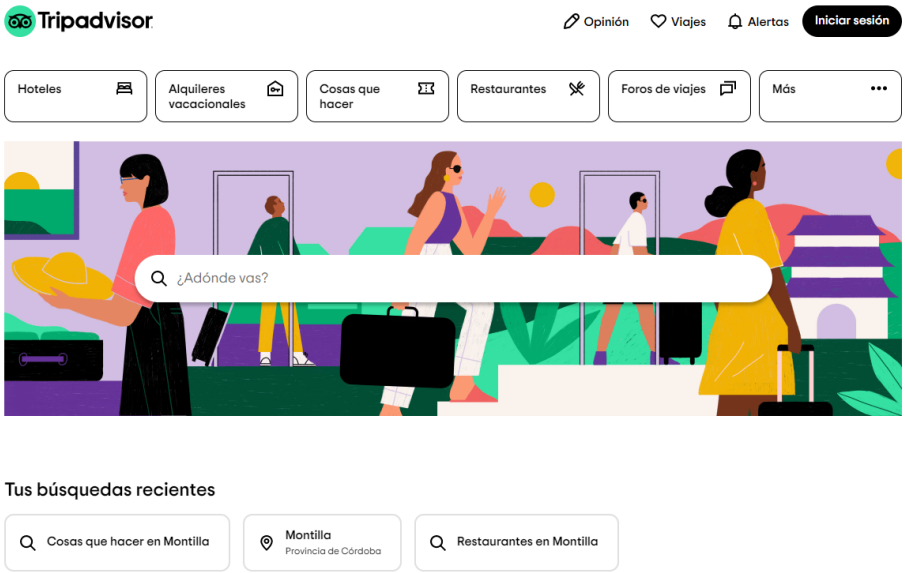


Figura 1 Ejemplo interfaz inicial Tripadvisor

- **AroundMe**: Esta es una aplicación muy simple, que simplemente realiza una búsqueda radial de negocios filtrando por una etiqueta concreta, bar, banco, gasolinera etc... Pero no permite ni opinar ni observar valoraciones sobre estos negocios y además no proporciona mucha información sobre los mismos.

    - **Fortalezas**: La simpleza de su interfaz y su baja curva de aprendizaje.
    - **Debilidades**: La información entre diferentes negocios es inconsistente, puedes entrar a buscar un negocio y no encontrar apenas información sobre el mismo, de algunos no se provee más información que la localización y no hay una estructura general que permita qué información vas a encontrar sobre un negocio al buscarlo en la aplicación.

Figura 2 Interfaz inicial AroundMe

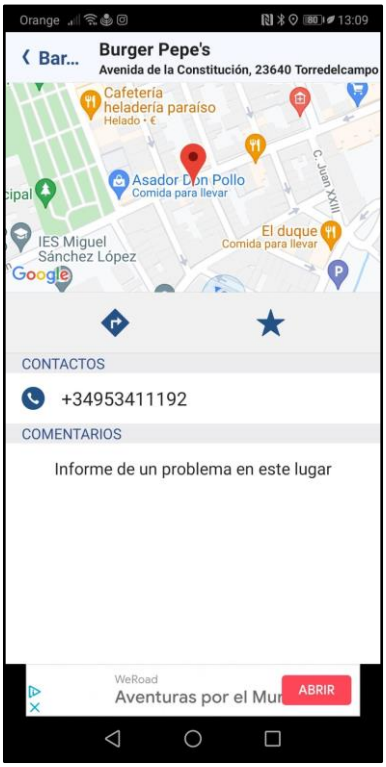


Figura 3 Ejemplo de negocio en AroundMe

- **Yelp:** Es una aplicación que permite localizar diversos negocios. Al igual que *AroundMe,* permite búsquedas radiales desde la ubicación del usuario así como el filtro por etiquetas según el tipo de negocio a buscar. Ofrece una gran variedad de categorías por las que buscar y permite realizar y comparar reviews de otros usuarios
    - **Fortalezas**: El buen diseño de su interfaz, la posibilidad de permitir a los usuarios tener una foto de perfil que los identifique ayuda a humanizar el proceso de valoración, permitiendo un enfoque más personal en las mismas, ya no son valoraciones de cuentas sin imagen, el añadir una foto de perfil a este tipo de perfiles es una sutil pero interesante característica que ha sido considerada para el desarrollo de la aplicación de este proyecto. También permite la posibilidad de contactar directamente con los negocios y la posibilidad de buscar ofertas, así como de ver los menús que estos negocios ofertan.
    - **Debilidades**: La aplicación en su conjunto es muy buena y aunque no sean debilidades como tal, se ha observado que en lo que respecta a la categoría de compras, la web toma un carácter muy genérico, en este proyecto se permite evaluar negocios y lugares que no se encuentran disponibles en la aplicación, como sucursales de bancos, pequeñas tiendas,etc…

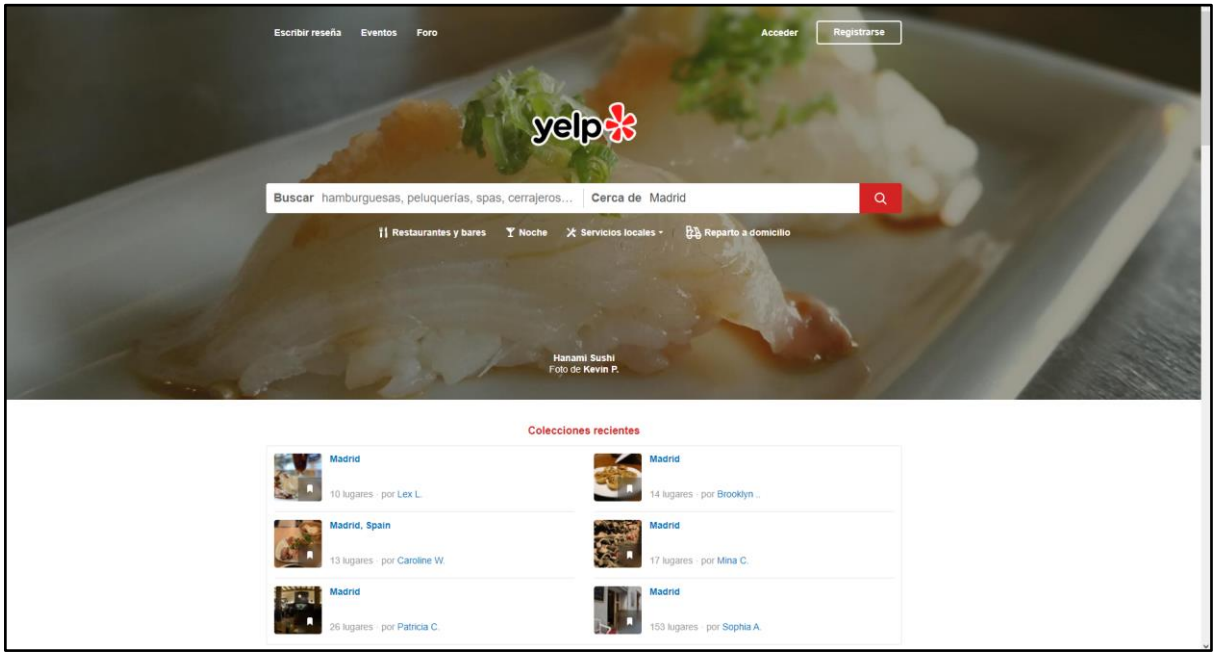


Figura 4 Página inicial de Yelp

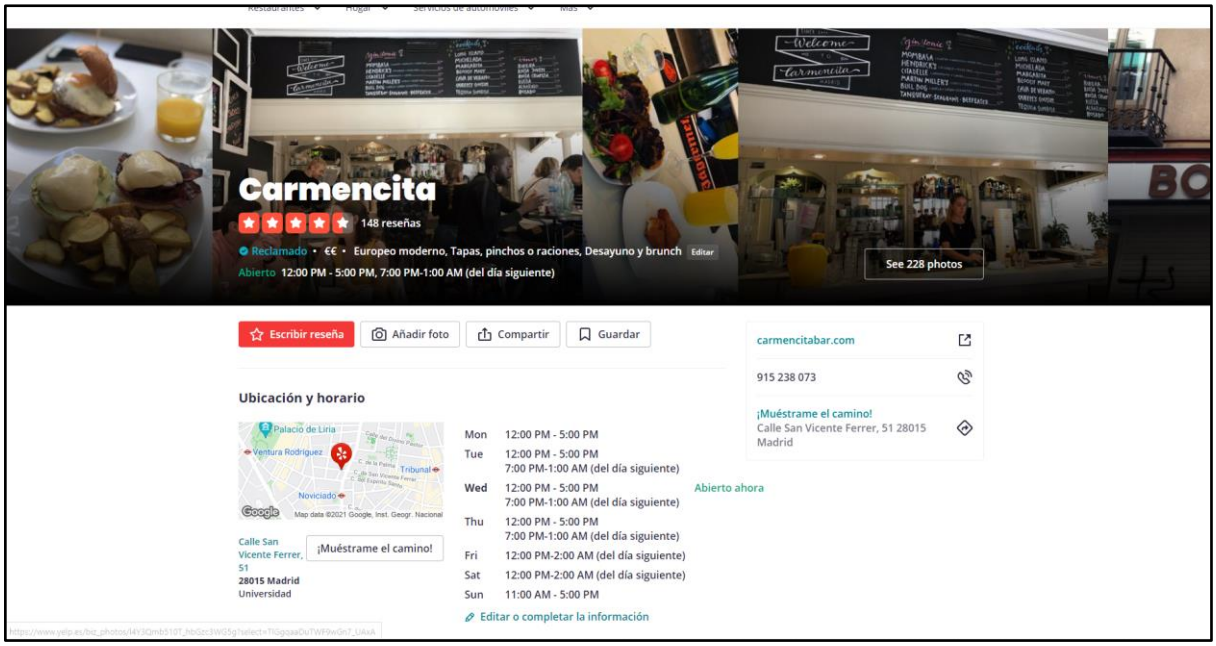


Figura 5 Ejemplo de negocio en Yelp

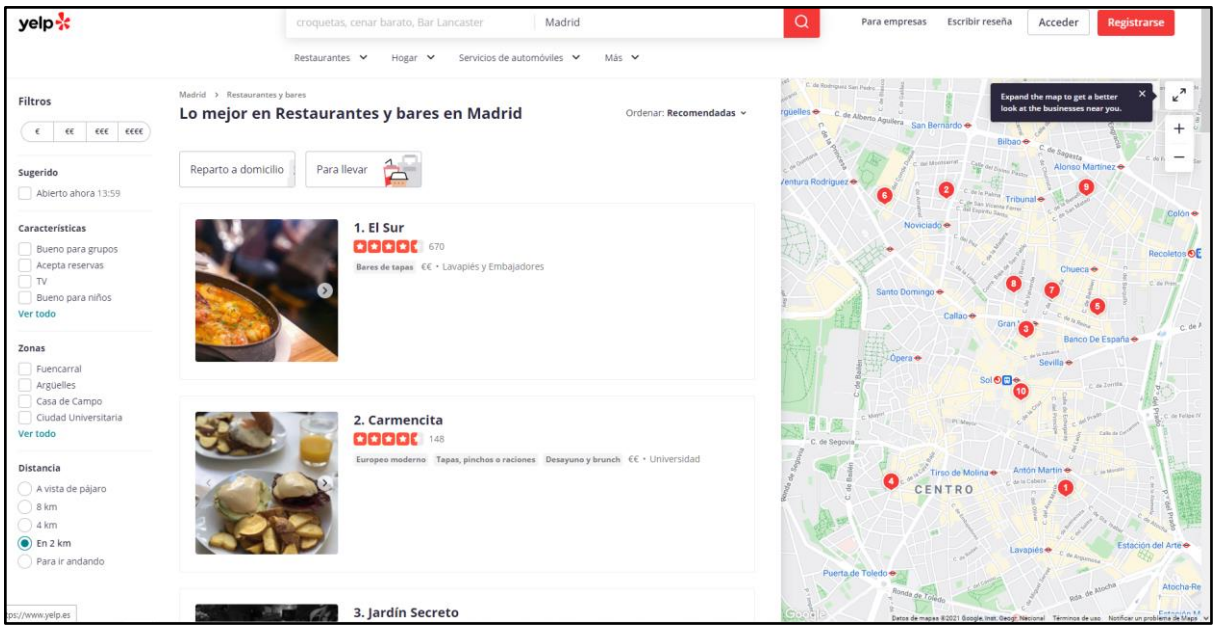


Figura 6 Ejemplo de búsqueda en Yelp

2.1.2 Selección de tecnologías

Para el desarrollo del sistema propuesto, se considera que tanto una aplicación web, como una aplicación móvil serían soluciones adecuadas, ya que son las 2 principales plataformas en las que se utilizan las redes sociales hoy en día, de acuerdo al siguiente estudio [1]. Según dicho estudio, pese a que el 83 % del uso de las redes sociales se haga en dispositivos móviles y sólo un 15 % use de forma exclusiva el ordenador, encontramos que incluso en ese 83% hay un 33% que también usa el ordenador para acceder a redes sociales, ya que solo aproximadamente el 50% de los usuarios accede a sus redes sociales exclusivamente a través de un dispositivo móvil (Telemedia ONLINE), por lo que el desarrollo de una solución web para la red social está más que justificado.

Algunas de las tecnologías consideradas para el desarrollo y sus características principales son las siguientes:

- PHP: Creado en 1994 es un lenguaje de programación que permite ejecutar código de back-end en documentos HTML, es un lenguaje fácil de aprender y ampliamente usado en el mundo empresarial hoy en día, según una encuesta [2]el 79,2 % de los desarrollos de back-end están programados en PHP. Su flexibilidad y facilidad de uso convierten a PHP en un candidato natural para los desarrollos de back-end para los cuales existen gran cantidad de frameworks siendo algunos de los más populares: Laravel, CodeIgniter y Symfony. [3].

- Angular: Angular es un framework open-source orientado al desarrollo del front-end para aplicaciones web SPA (Single-Page-Applications) mantenido por Google. Fue desarrollado por primera vez en 2010 como AngularJS y se ha ido actualizando casi anualmente a todos los niveles, arquitectura, sintaxis y rendimiento para poder competir con el resto de frameworks. Angular es un TypeScript-based framework, lo que significa que no trabaja con JavaScript directamente sino que utiliza TypeScript un lenguaje de programación más sencillo de entender y mantener que en la compilación se transforma en JavaScript para poder ser ejecutado por los navegadores. Es utilizado por empresas como Google y Microsoft a nivel de producción por lo que su utilidad y su potencia quedan más que demostradas.

- Spring: Es un framework desarrollado por Rod Johnson publicado bajo la licencia Apache 2.0 en Junio del 2003. Está programado en Java y una de sus características más positivas es su estructura modular, lo cual permite integrar nuevos elementos en la aplicación sin grandes complicaciones, siendo una gran opción para desarrollos orientados a microservicios. Además de su estructura modular el trabajo de la comunidad que apoya Spring ha permitido el desarrollo de diferentes proyectos que potencian sus posibilidades como framework, como menciona Kumar Chandrakant [4] : "*What makes Spring much more valuable is a strong ecosystem that has grown around it over the years and that continues to evolve actively. These are structured as spring projects which are developed on top of the Spring framework*.".

A parte de las tecnologías mencionadas con anterioridad algunas herramientas que se han tenido en cuenta para su posible uso en el desarrollo de la solución final son las siguientes:

- OpenStreetMaps: Creado en 2004 por Steve Coast, es un proyecto colaborativo que busca crear un mapa gratuito y editable del mundo, ofrece información proporcionada por su propios usuarios, a través de diversos medios (Dispositivos GPS, fotografías aéreas,etc…) (Neis and Zipf) y es usado para la creación desde mapas a papel como electrónicos, geolocalizar direcciones y lugares famosos y planificar rutas. En su página de wikipedia podemos encontrar algunos de sus usuarios más famosos: " *Prominent users include Facebook, Wikimedia Maps, Apple, Microsoft, Amazon Logistics, Uber, Craigslist, Snapchat, OsmAnd, Maps.me, Geocaching, MapQuest Open, JMP statistical software, and Foursquare."* ("Open Street Map").

- GoogleMaps: Desarrollado en C++ por los hermanos Lars y Jens Eilstrup Rasmussen , Noel Gordon y Stephen Ma y adquirido en octubre del 2004 por Google [5]. Es la aplicación de navegación más utilizada por los usuarios de smartphones, siendo usada por un 67 % de los encuestados según una encuesta realizada por The Manifest [6]. Google Maps provee una API sobre la que se puede trabajar para obtener información que va desde rutas entre lugares a horarios de aperturas de negocios, flujo de clientes según la hora, localización geográfica del usuario, entre muchas otras.

- Docker: Según su artículo de wikipedia "**Docker** es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos." [7]. A pesar de que Docker es un proyecto relativamente nuevo, creado en 2013, está siendo usado cada vez de forma más extensa en numerosos despliegues a nivel de producción por empresas de todos los tamaños, su facilidad para desplegar aplicaciones de forma segura, compacta y modular le ha permitido ganar una buena reputación pese al poco tiempo que lleva en funcionamiento.

2.2 Propuesta de solución

Como solución al problema planteado se pensó en diseñar una red social que permita obtener información de negocios cerca del usuario, utilizando algún sistema de filtro por etiquetas para clasificar los negocios y usando la información almacenada de los mismos puede devolver los negocios que interesan al usuario en un área a su alrededor. Considerando esta idea inicial, se seleccionaron las siguientes tecnologías:

Para el apartado del back-end, dado que la aplicación va a estar orientada a los microservicios se seleccionó Spring como núcleo para el desarrollo. La experiencia previa en este framework y su idoneidad para el modelo de aplicación que se quería desarrollar junto con la facilidad de poder utilizar JSP para el desarrollo del front-end de forma unificada usando Spring Boot JSP, así como las facilidades que provee para desarrollar y mantener APIs sólidas lo hicieron una elección fácil.

Para la obtención de la ubicación de los negocios en un mapa inicialmente se pensó en utilizar OpenStreetMaps, ya que es un servicio gratuito, pero debido a que se consideró más importante la información que me proporcionaba Google sobre estos y además se descubrió que una gran cantidad de pequeños negocios no se encontraban en OpenStreetMaps (los cuales son el objetivo de esta aplicación), esta opción fue rápidamente descartada, decantándose al final por el servicio de Google Places y la API de Google Maps.

Para el despliegue de la aplicación en la web se hizo un primer contacto con la tecnología de contenedores (Docker) mencionada brevemente en una asignatura de la carrera y quedando muy interesado por las facilidades que provee y lo sencillo de su uso fue integrada en el proyecto. Posteriormente tras investigar plataformas en la nube en la que realizar nuestro propio despliegue, con el fin de realizar un desarrollo utilizando en la mayor medida de lo posible los servicios de Google se acabó decidiendo el usar los contenedores creados para desplegar la aplicación en el servicio de Google Cloud.

Aunque inicialmente se planeó delegar el desarrollo del front end a Spring utilizando archivos .jsp, finalmente se decidió incorporar alguna tecnología o framework nuevo con el que no hubiese trabajado previamente en la carrera, así que se decidió utilizar Angular. La estructura modular con la que Angular trabaja, lo fácil que resulta escalar la aplicación y crear e integrar nuevos elementos de forma sólida, junto con la capacidad que provee para construir SPAs y la posible integración en un futuro con otros frameworks para el front-end como Ionic o Cordova, que se descartó incluir por añadir demasiada complejidad a una tecnología con la que no se había trabajado previamente, fueron los factores decisivos que finalmente decantaron la elección de este framework.

2.2.3 Historias de usuario

Las historias de usuario son una explicación informal y simple de una funcionalidad en las metodologías ágiles Se redactan desde la perspectiva del cliente o usuario final, no entran en detalles ya que estos se concretan posteriormente con un lenguaje no técnico para ayudar al equipo de desarrollo a entender lo que se desarrolla y cómo mejora el producto final [8]

2.2.3.1 Historias de usuario estimadas

En una primera estimación inicial del proyecto se consideraron las siguientes historias de usuario. Cabe aclarar que estas no son las historias que abarca el proyecto sino que estas son las mínimas funcionalidades que una aplicación como la que se describe en este proyecto debería tener. En la planificación se ajustarán las historias de usuario para poder adecuarse a un desarrollo según el plazo establecido. Para su definición se utilizará la notación "Como X quiero Y para Z, donde X será un tipo de usuario, Y una acción y Z un objetivo". Cada historia de usuario tendrá un código de historia de formato CHXX donde XX serán 2 números para poder ser referenciada en otros apartados :

- **Crear negocio, CH01 :** Como dueño de un negocio quiero poder crear un perfil para mi negocio para así poder tener visibilidad online.
    - Criterios de aceptación:
        - No se puede registrar un negocio que repita el googleID de otro negocio existente
- **Registrar Usuario, CH02**: Como usuario quiero poder registrarme en la aplicación para poder opinar sobre los negocios que me interesen.
    - Criterios de aceptación:
        - El usuario debe proveer un email que no esté registrado en la base de datos, en caso de que el email ya esté registrado se le notificará con un mensaje
- **Ver perfiles negocios, CH03:** Como usuario quiero poder ver los perfiles de los negocios para obtener información sobre ellos.
    - Criterios de aceptación:
        - El usuario debe poder visualizar los perfiles de los negocios incluso si no ha iniciado sesión.
- **Iniciar sesión, CH04:** Como usuario quiero poder iniciar sesión en la aplicación web para poder acceder a mi información personalizada.
    - Criterios de aceptación:
        - El usuario tiene que ser notificado de alguna forma de que el inicio de sesión ha sido satisfactorio, por ejemplo redireccionándolo a su perfil.
        - La página de inicio de sesión debe contener un enlace a la de registro para los usuarios que no tengan una cuenta registrada.

- **Tablón de usuario, CH05**: Como usuario quiero poder mostrar mis comentarios en un tablón propio para compartir mis opiniones sobre lugares que he visitado que me han parecido interesantes.
    - Criterios de aceptación:

        - El perfil de usuario tiene que actualizar automáticamente todos los comentarios que el usuario escriba y las puntuaciones que estos reciban
- **Personalizar perfil, CH06:** Como usuario quiero poder personalizar mi perfil para poder compartir algo de información sobre mi con otros usuarios.
    - Criterios de aceptación:
        - El usuario tiene que poder actualizar su nombre y biografía
        - El sistema tiene que notificar al usuario que está en modo edición y no guardará los cambios hasta recibir confirmación
- **Seguir negocio, CH07:** Como usuario quiero poder seguir a otros negocios para poder seguir al corriente de los eventos que realicen.
    - Criterios de aceptación:
        - El sistema tiene que permitir dejar de seguir a los negocios que ya se siguen
- **Comentar negocios, CH08:** Como usuario quiero poder poner comentarios sobre los lugares que visito para poder expresar mi opinión sobre ellos.
    - Criterios de aceptación:
        - El comentario tiene que tener una longitud máxima en caracteres
- **Personalizar información negocio CH09:** Como dueño de un negocio quiero poder personalizar la información en la aplicación sobre mi negocio si quiero que esta sea diferente de la obtenida por defecto de la obtenida de Google para poder transmitir mejor la esencia de mi negocio a los usuarios.
    - Criterios de aceptación:
        - El dueño de negocio tiene que poder ajustar los horarios en diferentes formatos
- **Puntuar comentarios, CH10:** Como usuario quiero poder puntuar los comentarios de otros usuarios para mostrar que estoy de acuerdo con lo que han dicho.
    - Criterios de aceptación:
        - Si un usuario intenta puntuar un comentario que ya ha puntuado se elimina la puntuación anterior
- **Suscribirse a un negocio, CH11:** Como usuario quiero poder suscribirme a un negocio pagando una tarifa para recibir descuentos en sus artículos o productos.
    - Criterios de aceptación:
        - Si un negocio es eliminado del sistema la suscripción tiene que cancelarse automáticamente
- **Borrar comentarios, CH12:** Como usuario quiero poder borrar un comentario que escribí previamente para que otros usuarios no lo vean.
    - Criterios de aceptación:
        - Un usuario no puede visualizar el botón de eliminar de comentarios que no sean suyos

- **Puntuar negocio, CH13:** Como usuario quiero poder puntuar un negocio para poder compartir mi opinión sobre ellos.
    - Criterios de aceptación:
        - Si un usuario puntúa un negocio que ya había puntuado previamente su puntuación anterior se elimina y se sustituye por la nueva.
- **Importar fotos de negocios, CH14**: Como dueño de un negocio quiero poder importar fotos de mi negocio desde google para facilitar a otros usuarios el identificarnos.
- **Listar negocios, CH15:** Como dueño de un negocio quiero poder acceder a una lista de mis negocios para poder acceder a ellos rápidamente.
    - Criterios de aceptación:
        - El usuario tiene que poder acceder a la creación de un nuevo negocio desde la misma vista para visualizarlos
- **Lista de favoritos, CH16:** Como usuario quiero tener una lista de lugares favoritos para poder acceder fácilmente a la información de los locales que más me interesan.
    - Criterios de aceptación:
        - En la vista se tienen que poder ver la puntuación de cada negocio así como el número de comentarios que tiene
- **Notificaciones de negocios, CH17:** Como usuario quiero poder recibir notificaciones de mis negocios favoritos para poder mantenerme informado sobre sus actividades y ofertas.
    - Criterios de aceptación:
        - Se mostrarán notificaciones tanto de los negocios que se siguen como de aquellos marcados como favoritos
- **Filtrar negocios, CH18:** Como usuario me gustaría poder filtrar los negocios a mi alrededor para localizar solo aquellos que me interesan.
    - Criterios de aceptación:
        - El filtro se podrá realizar por varios criterios, tipo de negocio, nombre, dirección, etc...
- **Localizar negocios, CH19:** Como usuario me gustaría poder ver los negocios alrededor de mi para poder conocerlos..
    - Criterios de aceptación:
        - El usuario podrá acceder directamente al perfil del negocio desde la misma vista.
- **Cambiar foto de perfil, CH20:** Como usuario quiero poder cambiar mi foto de perfil para poder ser reconocido fácilmente.
    - Criterios de aceptación:
        - Solo se aceptarán como fotos de perfil ficheros de tipo .png o .jpeg

2.2.4 Planificación de las historias de usuario

Una vez planteadas las anteriores historias de usuario hay que puntuar las historias, es decir, asignarle una estimación del esfuerzo que supone desarrollarlas, y descomponerlas en tareas concretas que se asignan a los desarrolladores en función del rol que cumplan en el desarrollo

de la aplicación. Estas tareas son muy variadas, desde escribir documentación, desarrollar interfaces o programar funcionalidades. Pero antes de descomponer una historia en tareas se debe determinar la prioridad de una historia en el conjunto de la planificación. También a cada historia se le debe asignar un valor en puntos de historia, estos puntos representan el esfuerzo que requiere implementar dicha historia y son una estimación que se extrae de la información previa que se posea sobre el desarrollo de aplicaciones similares, experiencia previa en el uso de frameworks o herramientas variadas. En el caso del uso de metodologías ágiles estas puntuaciones se asignan en grupo entre todos los integrantes del equipo para obtener una mayor precisión, ya que se tiene en cuenta la experiencia del conjunto del equipo de desarrollo y no solo de aquellos que realicen la planificación inicial. Dado que este TFG es un trabajo individual las estimaciones se han basado en la experiencia previa en aplicaciones web que he ido adquiriendo a lo largo de mis estudios en el grado.

Para establecer la prioridad en las historias de usuarios se utilizó el método MoSCoW [9]. Este método consiste en establecer la prioridad de las historias en 4 categorías, basándose en el valor que estas pueden añadir al producto final :

- **Must**: La historia de usuario es crítica para el funcionamiento de la aplicación y debe ser incluida cuanto antes
- **Should**: La historia da un gran valor añadido a la aplicación y aunque no es imprescindible es altamente prioritaria.
- **Could**: La historia proporciona un cierto valor añadido aunque no es muy relevante, su prioridad es media
- **Would**: La historia proporciona un valor añadido bajo o muy bajo, su prioridad es muy baja

2.2.4.2 Descomposición de las historias de usuario en tareas

Una vez establecido este sistema para clasificar las historias de usuario, se muestra su descomposición en tareas, su valor en puntos de historia, es decir, el esfuerzo necesario para completar la tarea y la prioridad que se ha asignado a cada una:

- **Crear negocio, CH01:**
    - Puntos: 2
    - Prioridad Must
    - Tareas:
        - Crear componente de Angular para crear el negocio
        - Añadir conexión con la API
        - Implementar funcionalidad
- **Registrar Usuario, CH02:**
    - Puntos: 2
    - Prioridad Must
    - Tareas:
        - Crear DAO para los usuarios y sus operaciones CRUD
        - Crear componente de Angular para crear el usuario
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Ver perfiles negocios, CH03:**

    - Puntos: 2
    - Prioridad Must
    - Tareas:
        - Crear componente de Angular para visualizar el negocio
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Iniciar sesión CH04:**
    - Puntos: 2
    - Prioridad Must
    - Tareas:
        - Crear componente de Angular para visualizar el negocio
        - Crear servicio de Angular para gestionar la sesión
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Tablón de usuario CH05**:
    - Puntos 1
    - Prioridad Must
    - Tareas:
        - Crear componente de Angular para visualizar el perfil del usuario
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Personalizar perfil CH06:**
    - Puntos: 1
    - Prioridad Should
    - Tareas
        - Añadir funcionalidad al componente del perfil del usuario en Angular para poder personalizar su información
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Seguir negocio CH07:**
    - Puntos: 1
    - Prioridad Should
    - Tareas:
        - Añadir funcionalidad al componente del perfil del usuario en Angular par poder seguir a otro usuario
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Comentar negocios CH08:**
    - Puntos: 2
    - Prioridad Must
    - Tareas:
        - Añadir funcionalidad al componente del perfil de negocio para permitir hacer comentarios
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Personalizar información negocio CH09:**
    - Puntos: 1

    - Prioridad Would
    - Tareas:
        - Añadir funcionalidad al componente del perfil de negocio para permitir su personalización
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Puntuar comentarios CH10:**
    - Puntos: 1
    - Prioridad Should
    - Tareas
        - Añadir funcionalidad al componente del perfil de negocio para permitir puntuar los comentarios registrados
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Suscribirse a un negocio CH11:**
    - Puntos: 1
    - Prioridad Could
    - Tareas
        - Añadir funcionalidad al componente del perfil de negocio para permitir a los usuarios el subscribirse
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Borrar comentarios CH12:**
    - Puntos: 1
    - Prioridad Could
    - Tareas
        - Añadir funcionalidad al componente del perfil del negocio, y del usuario para permitir a los usuarios el borrar un comentario
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Puntuar negocio CH13:**
    - Puntos:1
    - Prioridad Should
    - Tareas
        - Añadir funcionalidad al componente del perfil del negocio para permitir a los usuarios puntuarlo
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Importar fotos de negocios CH14**:
    - Puntos: 2
    - Prioridad Would
    - Tareas
        - Añadir funcionalidad al componente del perfil de negocio para permitir importar las imágenes desde google
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Listar negocios CH15:**

    - Puntos:1
    - Prioridad Should
    - Tareas
        - Crear componente en Angular para mostrar la lista de negocios
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Lista de favoritos CH16:**
    - Puntos: 1
    - Prioridad Could
    - Tareas
        - Crear componente de favoritos en Angular para mostrar la lista de negocios favoritos.
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Notificaciones de negocios CH17:**
    - Puntos: 2
    - Prioridad Could
    - Tareas
        - Crear DAO para las notificaciones con sus operaciones CRUD
        - Crear componente para las notificaciones en Angular
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end

- **Localizar negocios CH18:**
    - Puntos: 2
    - Prioridad Must
    - Tareas
        - Crear componente en Angular para localizar negocios alrededor de la posición del usuario
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Filtrar negocios CH19:**
    - Puntos: 2
    - Prioridad Must
    - Tareas
        - Añadir filtro de negocios al componente de Angular de localizar negocios
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end
- **Cambiar foto de perfil CH20:**
    - Puntos: 1
    - Prioridad Would
        - Añadir funcionalidad al componente de perfil de usuario para permitir cambiar la foto de perfil por defecto
        - Añadir conexión con la API
        - Implementar funcionalidad en el back-end.

2.2.4.3 Perfil del equipo de desarrollo

Para el desarrollo de esta aplicación vemos 2 roles principales, que por la naturaleza del TFG ambos serán asumidos por una única persona:
Desarrollador web: sería el encargado de desarrollar aquellas funcionalidades relacionadas con el front-end, este tipo de desarrollador se encargará de todas aquellas tareas que impliquen crear o modificar componentes en Angular para la diferentes historias de usuario.
Programador: sería el encargado de realizar todas aquellas tareas que impliquen implementar la sección del back-end de las historias de usuario junto con el desarrollo de las diferentes partes de la API así como la integración de esta con el resto de servicios de la aplicación

2.2.4.4 Planificación inicial de las iteraciones

Dada la división en tareas anterior y los roles asignados para cada historia de usuario en cada iteración, a cada uno de los desarrolladores le corresponderá su respectiva labor, según ha sido especificado en los roles: desarrollo de los componentes de angular para los desarrolladores web y trabajar con el back-end y la API para los programadores Partiendo de esta base y según las puntuaciones dadas vemos que obtenemos un total de 33 puntos de historia, por lo que dividiremos el desarrollo de la aplicación en 6 iteraciones de un mes de duración en la que cada iteración abarca un total de 5 puntos, siendo la última iteración de 8 puntos.

A cada una de las iteraciones corresponden las siguientes historias y tareas según la priorización MoSCoW que hemos establecido:

- Iteración 1:
  - Montar esqueleto REST
  - Montar estructura Back-end
  - Crear DAOs[1] y DTO[2]s y implementar sus operaciones CRUD [3]
  - Crear Negocio CH01
  - Registrar Usuario CH02
  - Realizar documentación
- Iteración 2:
  - Ver perfiles negocios CH03
  - Iniciar sesión CH04
  - Tablón de usuario CH05
  - Personalizar perfil CH06
  - Realizar documentación
- Iteración 3:
  - Comentar lugares CH08

[1] DAO: Data Access Object: Componente software que actúa de interfaz entre una base de datos y una aplicación
[2] DTO: Data Transfer Object: Componente software cuya única funcionalidad es el contener información para transmitirla entre procesos
[3] CRUD: Create Read Update Delete: Conjunto de operaciones que se pueden realizar sobre una entidad en una base de datos

    - Seguir negocio CH07
    - Puntuar comentarios CH10
    - Puntuar Negocio CH13
    - Integrar base de datos online (1 punto)
- Iteración:4
    - Localizar negocios CH18
    - Filtrar negocios CH19
    - Listar negocios CH15
    - Integrar despliegue en la nube
    - Finalizar documentación

Para representar esta planificación se ha creado el siguiente diagrama de Gantt que desglosa estas iteraciones en el tiempo, cada una se ha planificado como 14 días de trabajo. Esta planificación se ha creado considerándose que un TFG tiene un tiempo de esfuerzo de trabajo estimado de aproximadamente 300 horas.

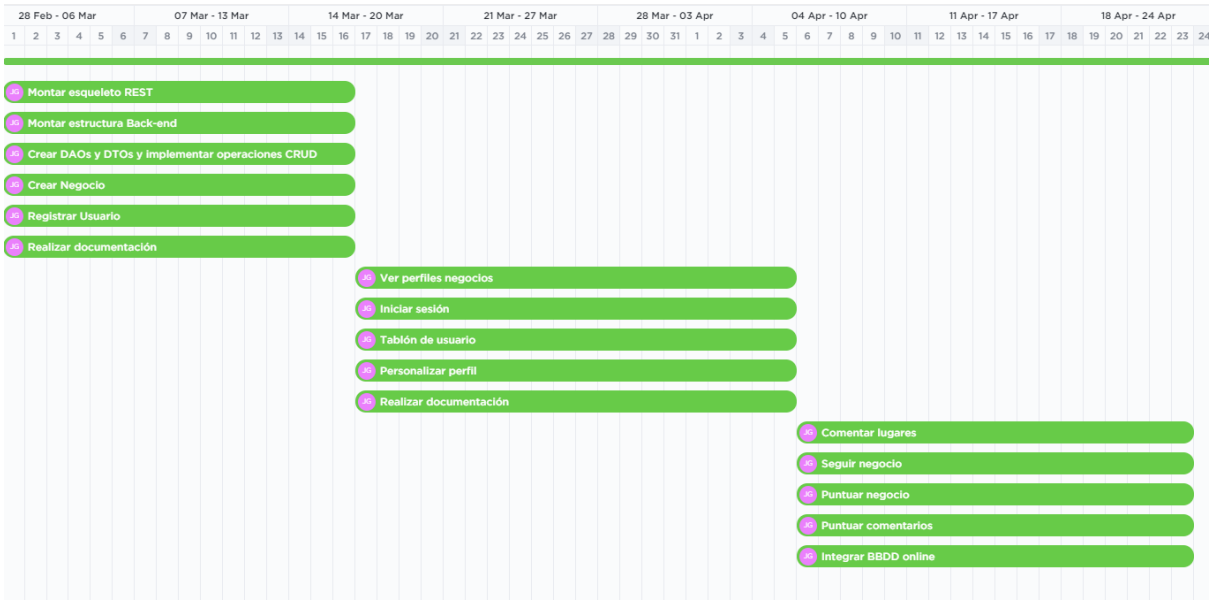


Figura 7 Diagrama de Gantt, planificación inicial, parte 1

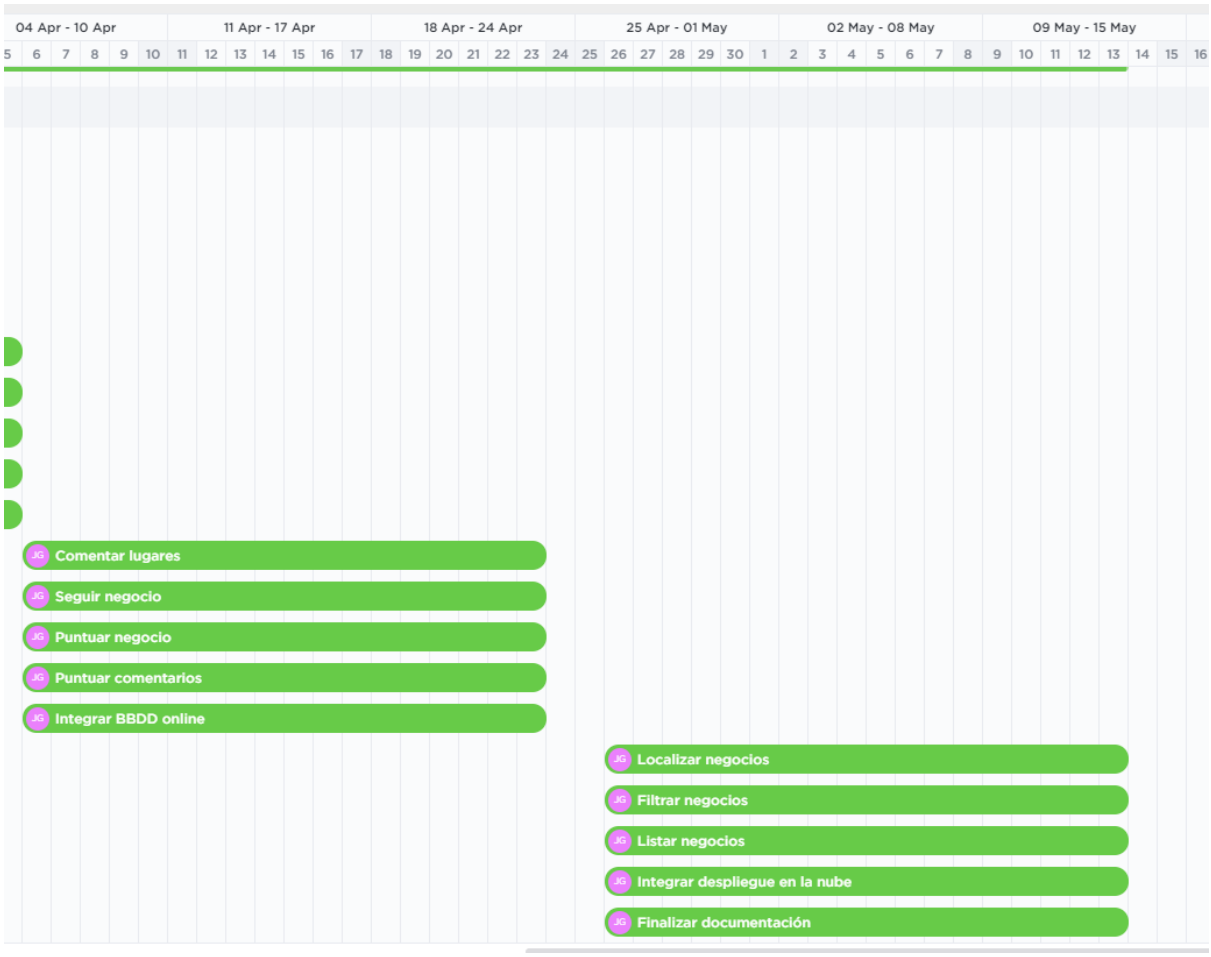


Figura 8 Diagrama de Gantt, planificación inicial, parte 2

2.2.5 Estimación de costes

Una vez desarrollada la planificación temporal del proyecto se pueden estimar los costes de desarrollo de este, estos costes los desglosamos en costes materiales, salarios y costes de despliegue.

El salario medio de un analista en España en la hora de desarrollo de este trabajo es de 14,26 € / hora [13], estimando unas 8 h de trabajo diarias podemos calcular el coste:
4 iteraciones * 14 días * 8 horas * 14,26€ / hora nos da un total de 6.388,48 € en salarios.

El equipo usado durante el desarrollo tiene un valor de 1200 €, suponiendo una vida útil de 4 años hasta que el hardware quede obsoleto, podemos estimar el valor que pierde el equipo durante el tiempo de desarrollo es de un 25 % por año, un año tiene 365 días, lo que son aproximadamente 52 semanas, si el desarrollo dura 8 semanas observamos que nuestro equipo pierde un valor de 46,15 €.

Respecto a los costes de despliegue, se utilizó la calculadora de precios de Google para el servicio de Cloud Run, se establecieron como parámetros las mínimas CPUs posibles, a unas 1000 peticiones por mes para poder realizar pruebas y el coste estimado es de 0 dólares por lo que es despreciable.

Para la documentación y diseño de los diagramas se utilizará el software Visual Paradigm en su versión de uso académico que gracias a la licencia proveída por la Universidad de Jaén será de coste 0.

A su vez se deben mencionar las licencias de uso de otros softwares de desarrollo que se utilizarán como serán Apache Netbeans y un editor de texto avanzado para el desarrollo de las vistas de Angular como puede ser Atom, sin embargo ambas licencias son de uso gratuito por lo que su coste para el proyecto será de 0 €.

Con todo esto podemos afirmar que el coste de desarrollo de la aplicación finalmente sería de unos 6434,63 €.

Concepto	Coste
Salario	6.388,48 €
Amortización equipo desarrollo	46,15 €
Coste despliegue Cloud Run	0 €
Coste peticiones Google Maps	0 €
Coste de licencias software de desarrollo y documentación adicionales	0 €
Importe total	**6434,63**

2.2.6 Modelo de dominio

Para el modelo de dominio se ha desarrollado el siguiente diagrama inicial.

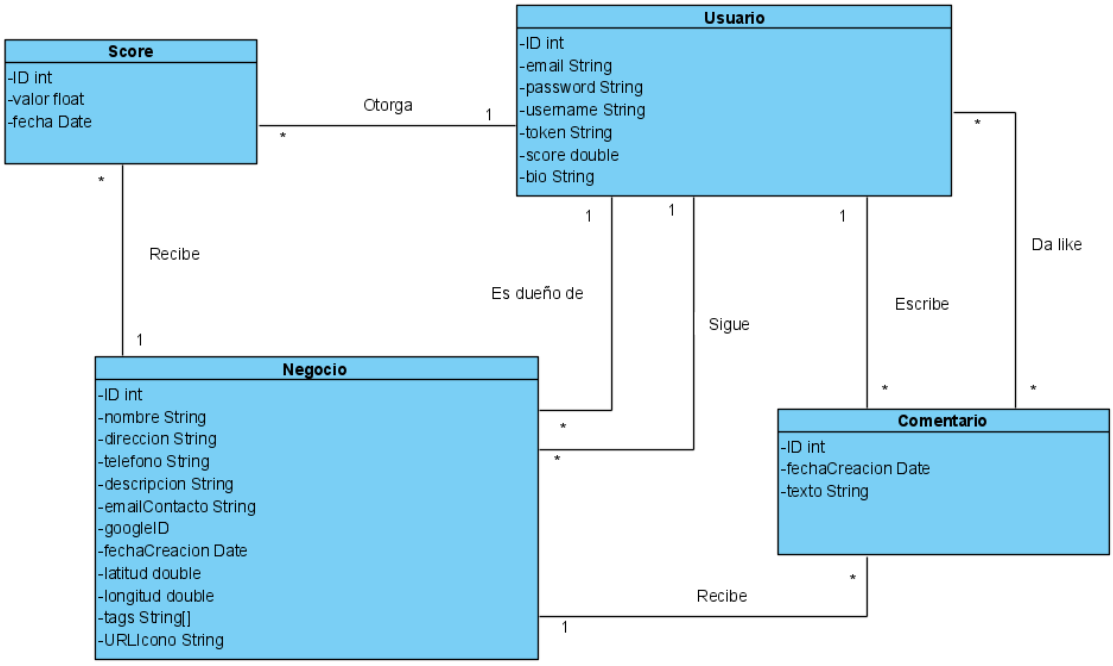


Figura 9 Diagrama de modelo de dominio

Para la mejor comprensión del anterior diagrama es conveniente atender a las siguientes aclaraciones:

La entidad *Score* surge para concretar una de las relaciones entre *Usuario* y *Negocio*, concretamente la que refiere a la puntuación que los usuarios otorgan a sus negocios. Esta relación 1 a muchos se refleja en el diagrama de tal forma que un usuario otorga muchas puntuaciones (o Score) a diferentes negocios y un negocio recibe múltiples puntuaciones de diferentes usuarios.

Un usuario puede seguir a múltiples negocios, los cuales a su vez pueden ser seguidos por múltiples usuarios, pero aunque un negocio solo puede tener un dueño, un usuario puede ser dueño de múltiples negocios.

Al igual que la entidad Score, Comentario surge para concretar elementos de una relación entre Usuario y Negocio, que sería una relación llamada “Comenta”, que en el diagrama finalmente se acaba fragmentando en Comentario y sus 2 relaciones con Negocio y Usuario, Recibe y Escribe respectivamente. *Comentario* permite almacenar detalles sobre esta relación. La cardinalidad de esta relación sería una muchos a muchos que al ser utilizar una clase de asociación se convierte en dos relaciones 1 a muchos, como se indica en el diagrama.
Por último la relación “Da like” corresponde a la capacidad de los usuarios de dar me gusta a un comentario en concreto, su carácter de muchos a muchos viene de que un usuario puede

dar me gusta a múltiples comentarios y un comentario puede recibir me gustas de múltiples usuarios.

Para la entidad Usuario cabe destacar el atributo token, que sera una cadena de caracteres generada aleatoriamente cuando se registre al usuario en el sistema, será única para cada usuario y se utilizará para validar las diferentes operaciones del mismo una vez halla iniciado sesión en el sistema.

A su vez la entidad Negocio tiene una serie de atributos para ayudar en su localización geográfica a través de *Google Places* estos atributos son, su latitud, su longitud y su identificación de *Google o Google ID* con esta información y el nombre del local podemos localizar de forma unívoca cualquier negocio que se encuentre en las bases de datos de *Google.*

3 Diseño

A continuación se presentan los diferentes tipos de diagramas empleados durante el diseño así como algunas aclaraciones y explicaciones

3.1 Diagrama E-R

Para el modelo relacional para la base de datos se presenta el diagrama de la ilustración 10. Este diagrama se derivará de las entidades mostradas anteriormente en el modelo del dominio y su generación será automática ya que en el sistema se ha decidido usar un sistema de ORM[4] como es JPA[5]. JPA permite mediante anotaciones establecer restricciones en las entidades que luego se reflejan de forma automática en la base de datos. Por ejemplo con una etiqueta @Id sobre un atributo podemos identificarlo automáticamente como la clave principal para la tabla asociada a la entidad en cuestión. A su vez permite generar de forma dinámica las tablas que surgen de las relaciones muchos a muchos marcando atributos que sean listas con @ManyToMany así como establecer las relaciones @OneToMany y otras restricciones de tipo columna como @Colum(unique=true) para hacer que un atributo que no sea la clave principal no se pueda repetir en otra tupla. En el apartado de detalles de implementación se comentarán más aspectos sobre esta cuestión.

[4] ORM: Object Relational Mapping: Es un modelo de programación que permite el mapeo de estructuras de bases de datos relacionales sobre una estructura lógica de entidades para simplificar el desarrollo de las aplicaciones.
[5] JPA: Java Persistence API: Es la API de persistencia desarrollada para Java EE.

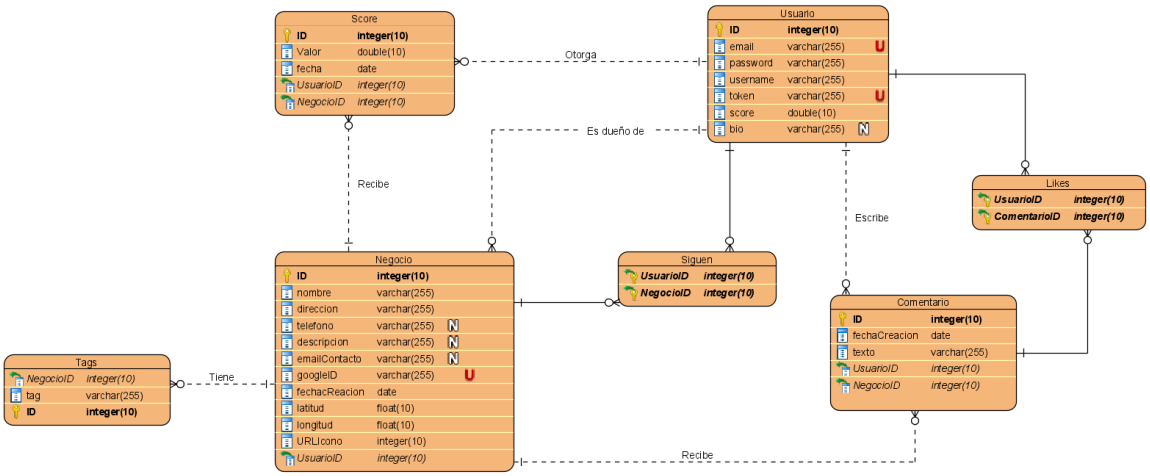


Figura 10 Diagrama E-R

Como podemos observar en el diagrama de la ilustración 10 la relación “Siguen” que se observaba en el diagrama de modelo de dominio de la ilustración 9 se ha transformado en una tabla con una clave primaria formada por la combinación de las clases primarias de las entidades *Negocio* y *Usuario*. Esta misma situación la encontramos en la tabla Likes. Todas estas tablas son generadas de forma automática por el ORM para preservar la estructura lógica del diseño, que como podemos observar en cada una de sus tablas, se encuentra en tercera forma normal para evitar la redundancia de los datos, ya que los atributos de cada tabla que no son parte de la clave primaria dependen directamente de esta.

3.2 Diagrama de Clases

Para el sistema propuesto se han diseñado dos partes diferenciadas, el back-end y el front-end, de acuerdo con la arquitectura cliente/servidor. Con sendos diagramas de clases que se indican en las figuras 11 y 12 daremos algunos detalles del diseño de cada uno de ellos

3.2.1 Diagrama de Clases Back-End

El diagrama de la ilustración 11 corresponde al back-end de la aplicación, no obstante, para facilitar su comprensión se han omitido algunos elementos redundantes. En primer lugar, solo aparece reflejado el DAO y DTO de la entidad Negocio, aunque en el sistema final existiría otros para las entidades Score, Comentario y Usuario. Además, estos DAOs poseen las operaciones básicas CRUD y los DTOs no poseen ninguna funcionalidad más allá de operaciones get y set, siguiendo el patrón DAO-DTO.

Para las funcionalidades principales del sistema se han dividido las responsabilidades entre 3 clases de servicios. *APIService* se encarga de la comunicación entre el back-end y el front-end, sirviendo como la clase que proporciona las funcionalidades de la API REST, como su propio nombre indica. *PlacesService* se encarga de la comunicación entre el back-end y los servicios de *Google Places.* Esta clase realiza todas las peticiones para obtener información de negocios basados en criterios de búsqueda por nombre, localización, etc…

Por último la clase *NegociosService* ocupa la mayor parte de la lógica del negocio, se encarga de gestionar la lógica de negocio detrás de las diferentes peticiones que se realizan a través de la API, a su vez hace uso del servicio *PlacesService* para conectar con Google y tratar así con la información de los negocios,

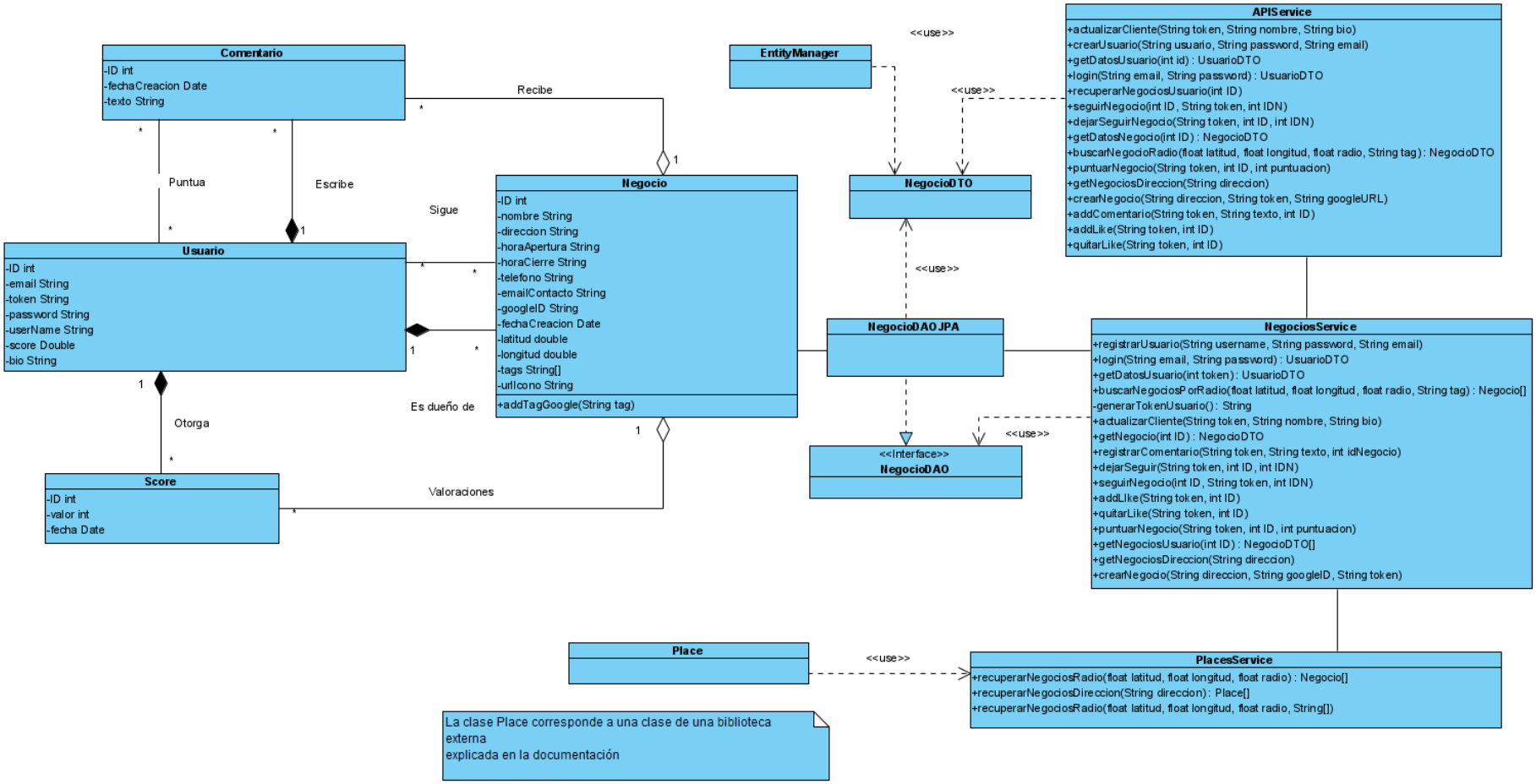


Figura 11 Diagrama de clases de la aplicación servidor (back-end)

3.2.2 Diagrama de clases front-end

Para entender el siguiente diagrama de clases en Angular hay que hacer previamente una serie de aclaraciones para entender mejor el funcionamiento del framework y el tipo de elementos del que dispone.

En Angular podemos encontrar diferentes tipos de elementos, los que se han usado en este proyecto y son los siguientes:

- Component: Esta clase es la unidad más básica con la que se construyen las aplicaciones de angular, consta de una página HTML, una clase de TypeScript para determinar el comportamiento del HTML y una página de css para su estilo
- Module: Esta clase sirve para organizar la aplicación, permite el importar y exportar diferentes bibliotecas y componentes de forma que constituye con ellos un “módulo”, para integrarlo en otras aplicaciones, suele ser el eje central de una aplicación angular
- Injectable: Son clases marcadas con la etiqueta @Injectable, permiten ser importadas para su uso en otros componentes a forma de bibliotecas de funciones

Hecha esta aclaración el diagrama de la ilustración 12 corresponde al apartado del front-end de la aplicación.

En él podemos destacar varias partes destacables. En primer lugar dos clases de servicios: *APIService* que se encarga de la conexión con la API del servidor y LoginService, que se encarga de gestionar los elementos referentes al mantenimiento de la sesión del usuario. Ambos servicios son usados por las clases que se derivan del AppModule, que sería la parte central de la aplicación de la que cuelgan los diferentes componentes que corresponden a cada una de las vistas de la aplicación. Al igual que en el diagrama de clases del backend, a pesar de que todos los componentes hacen uso del *APIService* para comunicarse con el servidor, solo se ha indicado esta relación en uno de ellos para simplificar el diagrama.

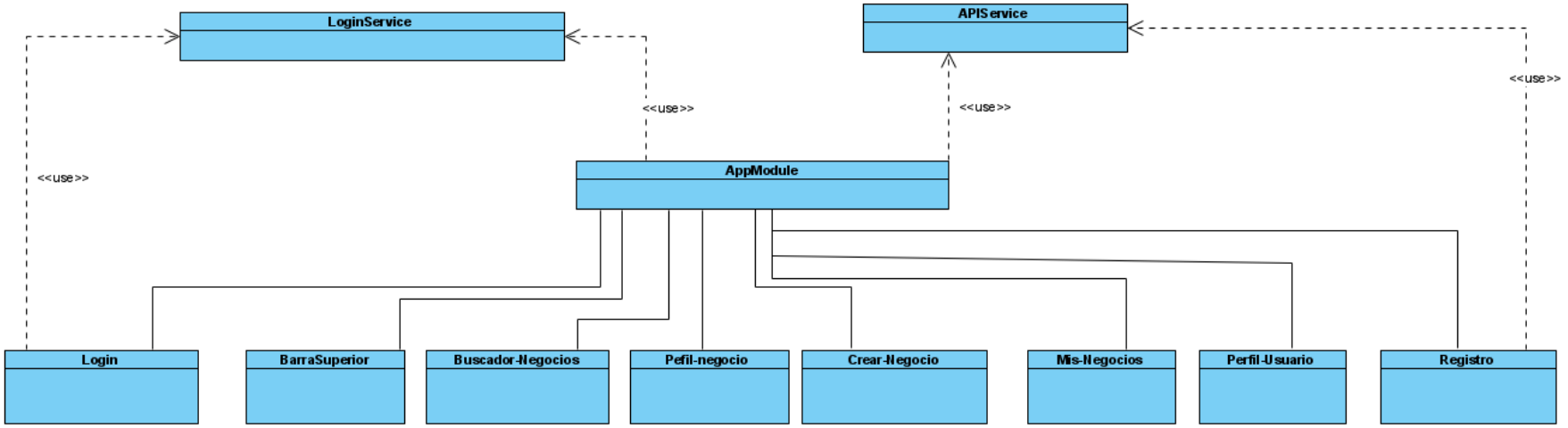


Figura 12 Diagrama de clases de la aplicación cliente (front-end)

3.3 Diagramas de secuencia

Para ilustrar el funcionamiento interno del sistema descrito en los apartados anteriores, se han desarrollado una serie de diagramas de secuencia que describen el comportamiento de las diferentes clases del sistema y cómo se comunican entre ellas y con otros elementos externos. Las ilustraciones 13 y 14 muestran cómo está diseñado el funcionamiento interno de la historia de usuario *Comentar Negocio.* En ellas podemos observar que los *Lifetimes*, es decir las columnas que representan a los componentes del sistema, presentan diferentes colores. Se ha elegido esta selección para facilitar su visualización y así entender a qué parte del sistema pertenece cada uno de sus elementos. El azul corresponde a la aplicación servidor o *back-end*, el amarillo a la aplicación cliente o *front-end y* el verde a los servicios de Google.

Se ha seleccionado esta historia de usuario ya que al abordarla en su totalidad se pueden observar varias partes que son comunes a otras historias de usuario, como el proceso de login o el buscar un negocio en un radio usando los servicios de Google.

Como se observa en el diagrama para crear un comentario primero el usuario debe iniciar sesión en el sistema, esto plantea nuestra primera situación alternativa reflejada en el diagrama de las ilustraciones 13 y 14, en caso de que el usuario no haya iniciado sesión no podrá publicar un comentario, por lo que para hacerlo debe entrar en la vista de *login.* En ella el usuario introducirá su usuario y contraseña y si la combinación de ambos corresponde a los de un usuario registrado previamente el sistema devolverá los datos del usuario. En caso contrario se notificará al usuario de que la combinación usuario-contraseña no es válida.

Tras haber iniciado sesión para localizar un negocio, se proporciona una dirección a la clase *Places Service* que se comunica con los servicios de Google para recuperar el negocio en esa dirección. Tras obtener este negocio, obtiene su id de Google y busca un negocio registrado con esa id en nuestro sistema y devuelve sus datos a la aplicación cliente, que muestra el perfil del negocio y desde este se permite al usuario poder crear comentarios. Para registrar un nuevo comentario un usuario proporcionará su token de identificación así como el texto del comentario en sí.

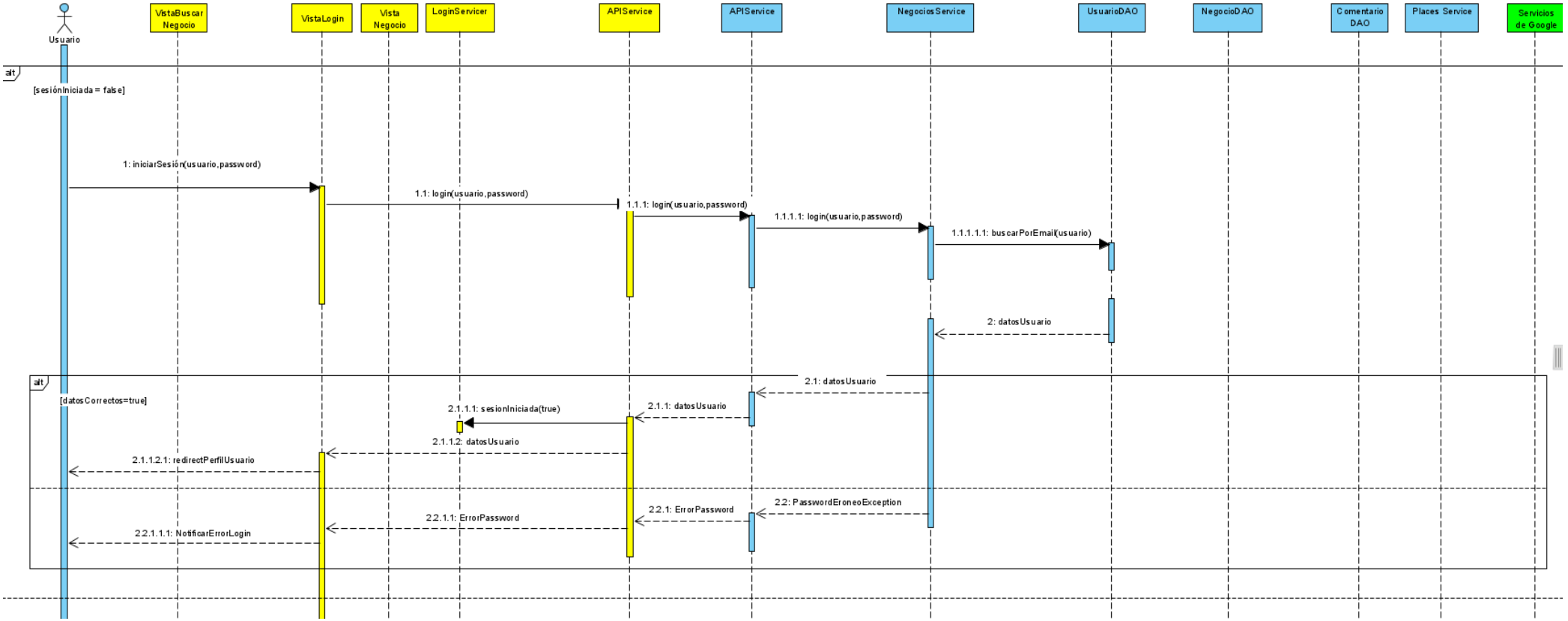


Figura 13 Diagrama de secuencia para la historia Comentar negocio, parte 1

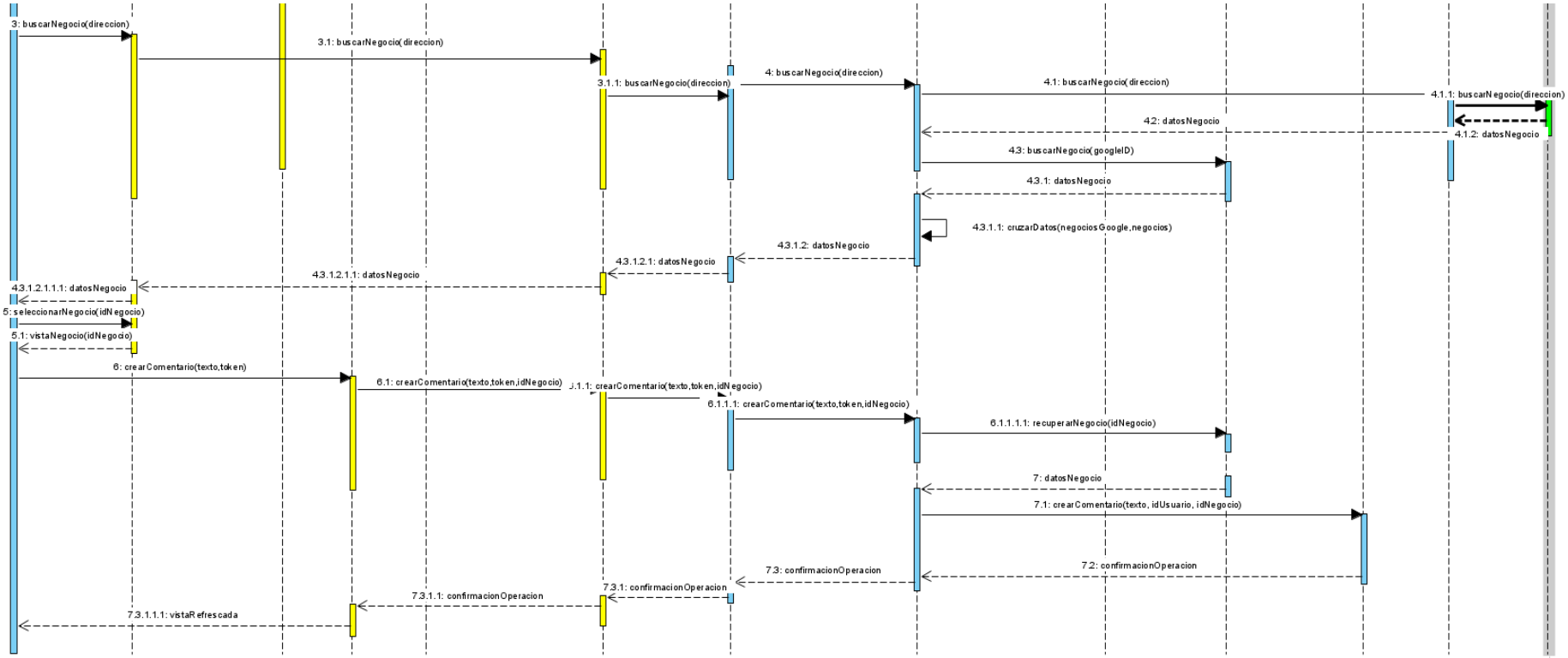


Figura 14 Diagrama de secuencia para la historia Comentar negocio, parte 2

Para el diseño del sistema también se ha considerado relevante el ilustrar cómo sería el proceso de registro de los nuevos negocios en el sistema usando los servicios de google, a tal fin se ha diseñado el diagrama de secuencia de las ilustraciones 7 y 8., en este diagrama se ha omitido el proceso de login, aunque es necesario ya que este proceso ya ha sido explicado con anterioridad, aun así es importante recalcar que para que un usuario pueda registrar un nuevo negocio primero tiene que iniciar sesión.

El proceso es el siguiente, el usuario entra en al vista de Crear Negocio donde se le solicitará una dirección y nombre para el negocio, esta información pasará por las diferentes partes del sistema indicadas hasta solicitar a los servicios de Google una lista de negocios en el área, tras esto el sistema devolverá una lista con datos básicos de estos negocios, como su nombre, tipo de negocio, etc… El usuario después seleccionará su negocio y el sistema solicitará los detalles del negocio a los servicios de Google y los almacenará para registrar un nuevo negocio. En caso de que ya exista un negocio cuyo googleID esté asociado a otro usuario, se notificará a través de un error, en caso contrario se redirigirá al usuario a la página del recién registrado negocio.

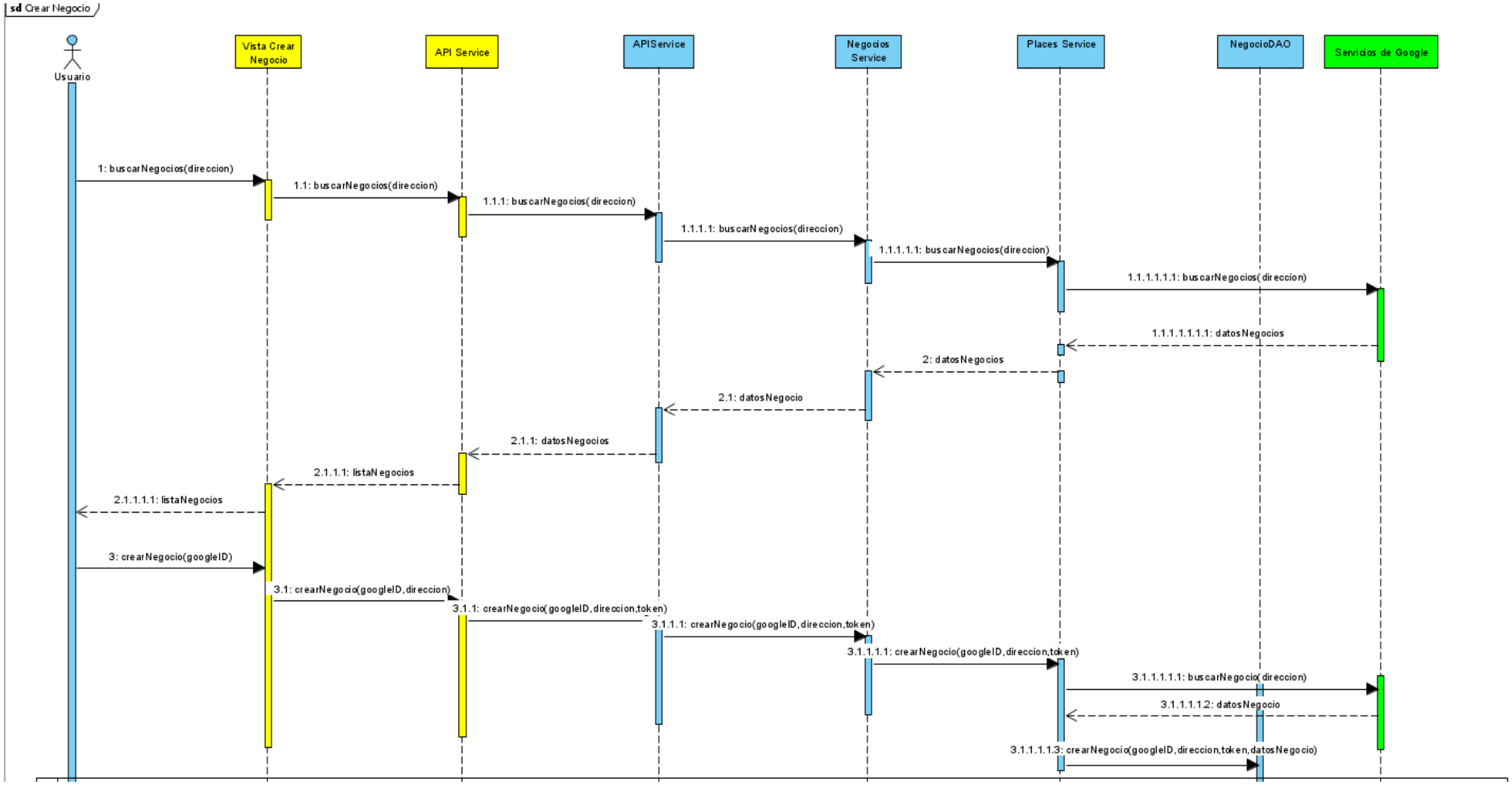


Figura 15 Diagrama de secuencia para la historia de usuario Crear Negocio, parte 1

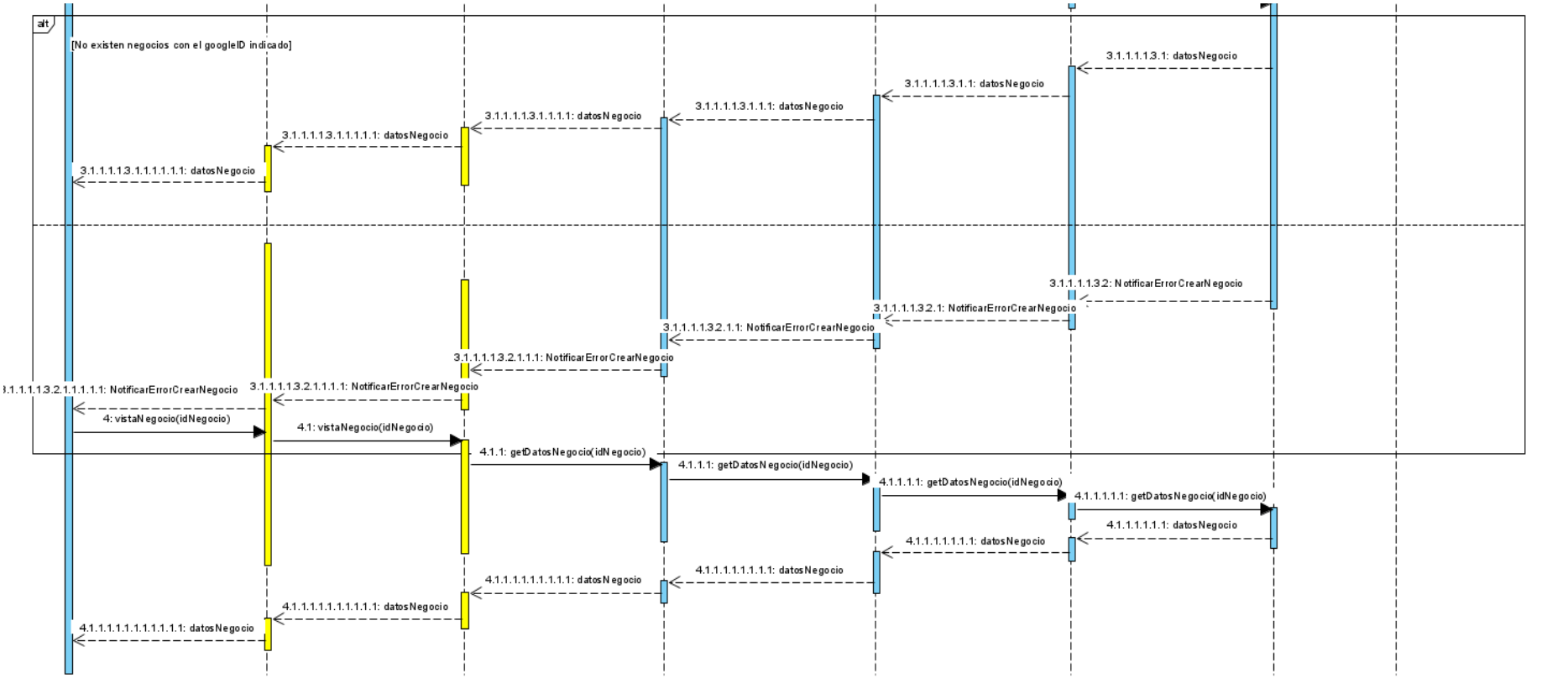


Figura 16 : Diagrama de secuencia para la historia de usuario Crear Negocio, parte 2

3.4 Diseño de la API

Para el diseño de la API REST se han clasificado los *endpoints*, es decir, cada una de las direcciones donde se pueden solicitar datos al servidor en función de la clase de la cual obtenemos el recurso. En este apartado sólo se mostrarán y comentarán brevemente algunos ejemplos de la documentación ya que el diseño completo de la API, desarrollado en www.swagger.io se adjuntará como anexo en la documentación.

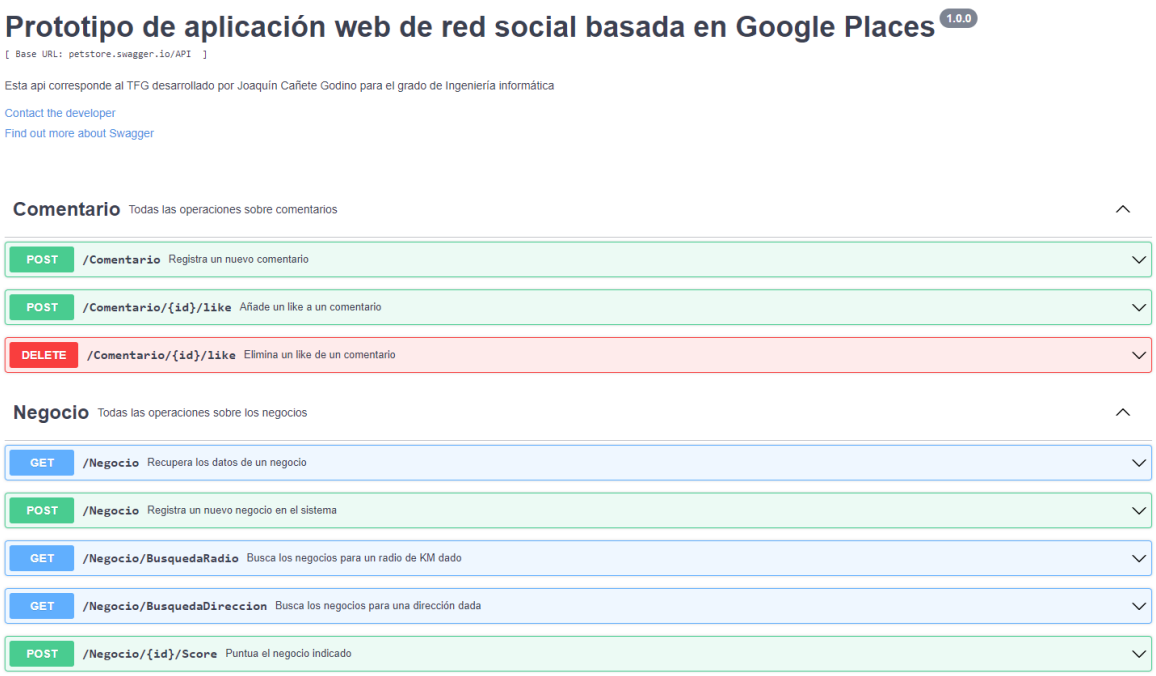


Figura 17 Diseño de la API

Como se puede observar en la ilustración 17 y con el fin de ilustrar brevemente el funcionamiento de la API podemos ver que para la entidad *Negocio* tenemos para una misma ruta */Negocio* dos operaciones diferentes en función del método con el que la realizamos. Con el método GET se obtiene los datos de un usuario, tras proveer el id que lo identifica como parámetro, para esta operación podemos recibir dos respuestas, un código 200 junto con los datos del negocio, si existe un negocio con el identificador que se ha proporcionado o un código 400 o BAD REQUEST con un texto descriptivo que indica que para el id proporcionado no existe ningún usuario registrado en el sistema. El método POST con la misma ruta sirve para registrar nuevos negocios en el sistema para ello se deberán proveer 3 parámetros, una dirección, el token del nuevo dueño del negocio y el googleID del negocio en los servicios de *Google Places,* para esta operación a diferencia de la anterior podemos obtener 3 códigos de respuesta, un 201 o CREATED en el caso de que el negocio se haya podido registrar con éxito, en caso contrario el sistema nos mandará un código 400 si no existe ningún usuario registrado en el sistema con el token proporcionado o un código 409 o CONFLICT si ya existe un negocio registrado para el googleID que se proporciona.

Para ambas peticiones en caso de que alguno de los parámetros se dejen en blanco el sistema devolverá un error 400 genérico debido a que todos los parámetros indicados, como se muestra en la documentación de swagger proporcionada en el anexo, son obligatorios.

3.5 Plan de pruebas

Para las pruebas se han de realizar las comprobaciones necesarias en cada una de las iteraciones para asegurar que los criterios de aceptación de las historias desarrolladas se cumplen tal y como están definidos en este documento. Para el desarrollo de estas pruebas se tomará un enfoque de caja negra. Para los criterios de aceptación de cada historia, yo mismo realizaré todas las pruebas a la finalización de cada una de las iteraciones y durante el desarrollo de las mismas. La situación ideal sería disponer de usuarios externos al desarrollo para realizar este tipo de pruebas pero al no disponer de un *product manager* o usuarios reales que puedan realizar las mismas, como sería recomendable en un desarrollo ágil, se ha decidido este enfoque. Para cada uno de los criterios de aceptación indicados en la descomposición de las historias hay que verificar tanto los casos positivos como los negativos, es decir, cuando la condición impuesta se cumple y cuando no.

A su vez, se tienen que comprobar las funciones de la API que se vayan desarrollando a lo largo del proyecto, para dicho fin se utilizará alguna herramienta de llamadas a APIs como puede ser Postman. Para cada *endpoint* de la API se debe comprobar que devuelva tanto los datos como los códigos de respuesta adecuados. Además, se deben realizar tantas pruebas por endpoint de la API como posibles códigos de error puedan presentar según el diseño que se ha presentado en el apartado 3.4.

4 Implementación y despliegue en la nube

En este apartado se va a mostrar la arquitectura general del sistema desarrollado, así como las diferentes tecnologías que se usan en cada una de sus partes. También se van a comentar una serie de detalles interesantes que deben ser tenidos en cuenta sobre la implementación del sistema para su posterior mantenimiento y actualización.

4.1 Arquitectura

La arquitectura del sistema es la mostrada en el diagrama de la figura 18 donde se muestran las tecnologías usadas en cada uno de sus componentes. Distinguimos 4 partes: los servicios de Google que proveen la información de los negocios originalmente, en particular los servicios de Google Places y Google Maps; la base de datos online donde almacenamos los datos de los usuarios utilizando SQL; el servidor desarrollado en Java y que utiliza Spring y, por último, otras bibliotecas externas de terceros que gestionan la información. La aplicación

cliente que es con la que interactúan directamente los usuarios está desarrollada con Angular y utiliza algunos elementos de Bootstrap.

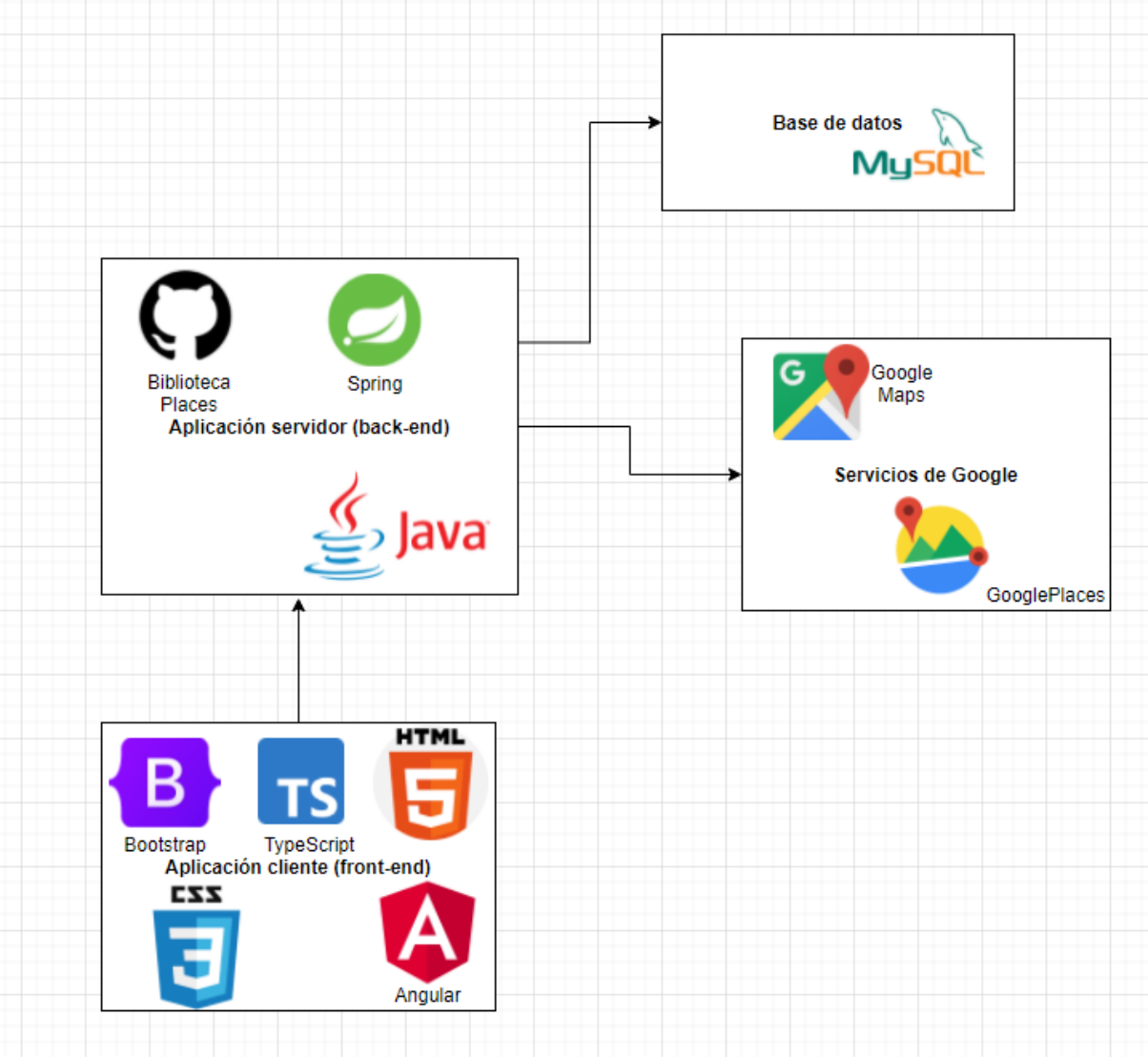


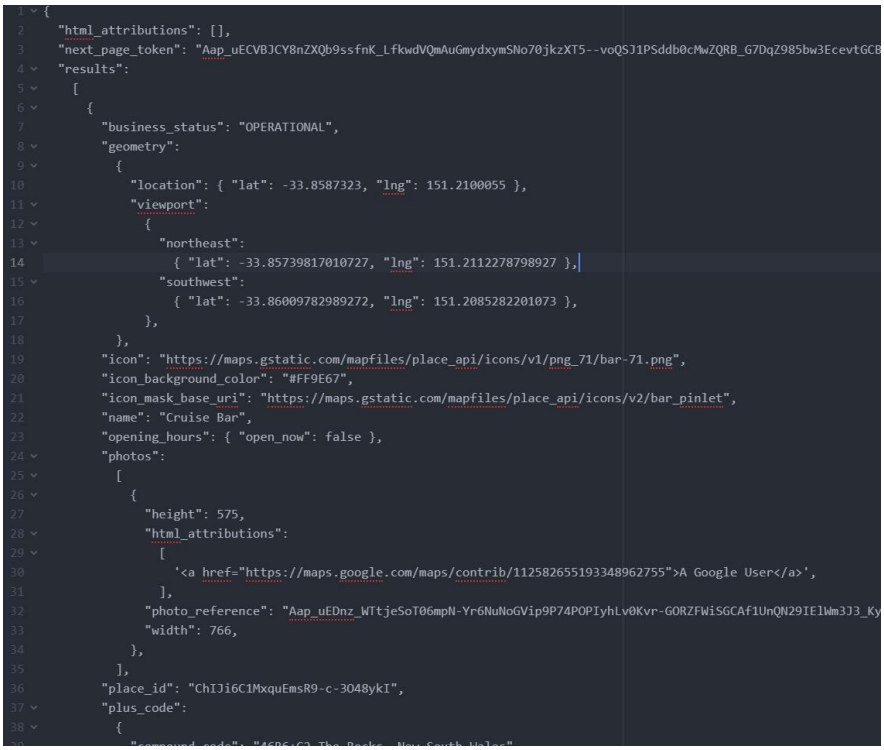
Figura 18 Diagrama de arquitectura del sistema

4.2 Detalles sobre implementación en el back-end

En este apartado se van a cubrir algunos detalles sobre la implementación de diferentes elementos del sistema que se han considerado relevantes de cara a la posible ampliación o mantenimiento del mismo, así como a facilitar un mejor entendimiento del sistema a un nivel más técnico y menos de diseño. Para ello, se explicará brevemente la acción que desempeñan las partes mostradas y se mostrará con fragmentos de código su funcionamiento interno.

4.2.1 Funcionamiento de la biblioteca externa *Places*

Uno de los elementos externos al sistema que hay que tener en cuenta en la implementación es la biblioteca Google Places que se usa en la clase de *PlacesService* en el backend. Esta biblioteca se ha seleccionado para simplificar el uso de la API de places de Google ya que manejar manualmente los objetos que genera la API puede ser bastante complejo dada la extensión de los mismos Un ejemplo de la respuesta lo podemos encontrar en la propia bibliografía de Google [10]**,** pero a fin de ilustrar brevemente la magnitud de una respuesta se provee la ilustración 19, aunque en la ilustración sólo se ven las primeras 38 líneas del JSON de ejemplo, este alcanza una longitud de 954 líneas, con una gran multitud de atributos complejos, lo que hace complicado trabajar con ella directamente.



```
{
  "html_attributions": [],
  "next_page_token": "Aap_uECVBJCY8nZXQb9ssfnK_LfkwdVQmAuGmydxymSNo70jkzXT5--voQSJ1PSddb0cMwZQRB_G7DqZ985bw3EcevtGCB
  "results":
    [
      {
        "business_status": "OPERATIONAL",
        "geometry":
          {
            "location": { "lat": -33.8587323, "lng": 151.2100055 },
            "viewport":
              {
                "northeast":
                  { "lat": -33.85739817010727, "lng": 151.2112278798927 },
                "southwest":
                  { "lat": -33.86009782989272, "lng": 151.2085282201073 },
              },
          },
        "icon": "https://maps.gstatic.com/mapfiles/place_api/icons/v1/png_71/bar-71.png",
        "icon_background_color": "#FF9E67",
        "icon_mask_base_uri": "https://maps.gstatic.com/mapfiles/place_api/icons/v2/bar_pinlet",
        "name": "Cruise Bar",
        "opening_hours": { "open_now": false },
        "photos":
          [
            {
              "height": 575,
              "html_attributions":
                [
                  '<a href="https://maps.google.com/maps/contrib/112582655193348962755">A Google User</a>',
                ],
              "photo_reference": "Aap_uEDnz_WTtjeSoT06mpN-Yr6NuNoGVip9P74POPIyhLv0Kvr-GORZFWiSGCAf1UnQN29IElWm3J3_Ky
              "width": 766,
            },
          ],
        "place_id": "ChIJi6C1MxquEmsR9-c-3O48ykI",
        "plus_code":
          {
```


Figura 19 Ejemplo respuesta API Google Places

Como vemos en el método recuperarNegociosRadio() mostrado en la siguiente ilustración

```java
public class PlacesService {
    @Autowired
    NegocioDAO_JPA negocioDAO;

    private GooglePlaces gPlacesService;


    public PlacesService(){
        gPlacesService= new GooglePlaces(Constantes.APIKEY);

    }

    public List<Negocio> recuperarNegociosRadio(float latitud,float longitud, float radio){
        List<Negocio> listaNegocios=new ArrayList<>();
        List<Place> listaLugares = gPlacesService.getNearbyPlaces(latitud,longitud,radio,GooglePlaces.MAXIMUM_RESULTS);
        for(int i =0;i<listaLugares.size();i++){
           String googleID=listaLugares.get(i).getGoogleUrl();
           Negocio n = negocioDAO.buscarGoogleID(googleID);
           List<String> listaTags=listaLugares.get(i).getTypes();
           if(n!=null){
               listaNegocios.add(n);
           }
        }

        return listaNegocios;
    }
```

Figura 20 Ejemplo de código de búsqueda radial con la biblioteca Google Places

Se usa esta biblioteca para hacer consultas a Google Places obteniendo directamente una lista de objetos de tipo *Place* que luego podemos usar para construir los objetos de tipo Negocio que son con los que trabaja la aplicación realmente. Estos objetos de tipo Place obtenidos poseen toda la información que Google permite recuperar sobre un negocio y podemos obtenerlos de varias formas, aunque en la aplicación no se han usado todas.

En el ejemplo de la ilustración 20 se realiza una búsqueda radial, es decir, dada una latitud, una longitud y un radio en kilómetros se buscan todos los negocios registrados en Google Places que pueden ser filtrados posteriormente según se quiera. Para dicha acción existen los “tags” que son una serie de cadenas de caracteres que permiten clasificar los negocios según su tipo ya sean bares, bancos, farmacias, teatros, etc...

Otra de las búsquedas que permite realizar esta biblioteca y que se ha usado en la aplicación es la búsqueda mediante una consulta o *Query;* es decir, nosotros damos una dirección como la buscaríamos por Google Maps y obtenemos los negocios que correspondan a la dirección dada. Un ejemplo de esta ejecución se ve en la ilustración 21

```java
public List<Place> recuperarNegociosDireccion(String direccion){

        List<Place> p =gPlacesService.getPlacesByQuery(direccion);

    return p;

}
```

Figura 21 Ejemplo de código de búsqueda por Query con la biblioteca Google Places

Todos los detalles referentes a esta biblioteca y al resto de sus funcionalidades pueden ser

encontrados en su página de GitHub [11]

4.2.2 Funcionamiento del atributo de seguridad o token

Otro detalle interesante sobre la implementación es el uso y del atributo token para la validación de los usuarios. Este token se genera en una clase auxiliar que sirve exclusivamente para su generación por lo que se omitió en el diagrama del diseño original.

```
public class TokenGenerator {
   private   KeyGenerator generador;

    public TokenGenerator(){
          try{
            generador= KeyGenerator.getInstance("AES");

        }catch(Exception e){
           System.out.println("El cifrado escogido  no esta disponible");
        }
    }

    public String generarKey(){
        String encodedKey;
        SecretKey key= generador.generateKey();

        encodedKey=Base64.getEncoder().encodeToString(key.getEncoded());
         char[] aux=encodedKey.toCharArray();
         char[] result = new char [encodedKey.length()];
         int u =0;
        for(int i =0;i<encodedKey.length();i++){
          if(aux[i]!='+'){
              result[u]=aux[i];
              u++;
          }
```

Figura 22 Ejemplo de código de la generación del atributo token

Como podemos observar en el código de la ilustración 22 se utiliza un *KeyGenerator* para generar una cadena de caracteres en *AES64* para identificar de forma unívoca a cada usuario. Para garantizar que el token de los usuarios no se repite ejecutamos la comprobación de la ilustración 23 durante el proceso de registro del usuario.

```
boolean tokenCorrecto=false;
 String token="";

while(tokenCorrecto==false){
 token= tokenGenerator.generarKey();
 Usuario prueba=usuarioDAO.buscarPorToken(token);
 if(prueba==null){
     tokenCorrecto=true;
 }
```

Figura 23 Ejemplo de código del registro de usuario

Este token se utiliza en todas las operaciones que resultan en la modificación de la información del usuario o de los negocios que posea. Como ejemplo podemos tomar la función de la ilustración 24 en la que se refleja la función de la API que permite a un usuario seguir un nuevo negocio. Aparte del id del usuario y del negocio, se debe pasar el token del usuario para verificar la operación. Para obtener el token, se debe realizar la operación de inicio de sesión con éxito.

```
@ResponseStatus(HttpStatus.OK)
@RequestMapping(value="/Usuario/{id}/Seguir",method=POST)
public void seguirNegocio(
@PathVariable int id,
@RequestParam String token,
@RequestParam int idN){
    this.core.seguirNegocio(id,token,idN);
}
```

Figura 24 Ejemplo de código de la función para seguir un negocio

Un ejemplo de cómo se usa el token en una llamada a la API de la aplicación servidor sería la mostrada en la ilustración 25. Por cuestiones de seguridad, el token es obligatorio para las operaciones que modifican los datos de un usuario

```
@ResponseStatus(HttpStatus.OK)
@RequestMapping(value="/Usuario", method=PUT)
public void actualizarCliente(
        @RequestParam(required=true) String token,
        @RequestParam String nombre,
        @RequestParam String bio ){
    core.actualizarCliente(token,nombre,bio);
}
```

Figura 25 Ejemplo de código de una función de la API

4.2.3 Funcionamiento de las interfaces DAO para el servicio de base de datos

Como se comentó en el apartado 3.1, la implementación de JPA en el lado del servidor para el mapeo automático de las clases y sus relaciones nos permite generar automáticamente las tablas de la base de datos según la necesitemos. Para mostrar el funcionamiento de este esquema se ha escogido la clase *Negocio* ya que incluye una gran cantidad de relaciones y restricciones interesantes que pueden servir para ilustrar las funcionalidades del framework

```
@Entity
public class Negocio {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int ID;
    private String nombre;
    private String direccion;
    private String horaApertura;
    private String horaCierre;
    private String telefono;
    private String emailContacto;
    @Column(unique=true)
    private String googleID;
    private Date fechaCreacion;
    private double latitud;
    private double longitud;
    @ElementCollection(targetClass=String.class)
    private List<String> tags;
    @OneToMany(mappedBy="objetivo")
    private List<Comentario> comentarios;
    @OneToMany(mappedBy="negocio")
    private List<Score> valoraciones;
    @ManyToMany
    private List<Usuario> seguidores;
    @ManyToOne
    private Usuario dueño;
    private String urlIcono;
```

Figura 26 Ejemplo de mapeado JPA de la entidad Negocio

Como podemos observar en la ilustración 26, para que una entidad sea mapeada automáticamente por JPA tenemos que definirla con la etiqueta @Entity. Tras esto debemos identificar un atributo como clave primaria con la etiqueta @Id. Esta clave primaria puede ser generada automáticamente a la hora de registrar una nueva tupla en la base de datos como se muestra en el ejemplo con la etiqueta @GeneratedValue, aunque esto último no es necesario y depende de cómo esté diseñado nuestro sistema. A su vez podemos establecer restricciones de tipo *Unique,* es decir, que un atributo de una tupla no puede repetirse en otras a pesar de no ser la clave primaria. Las relaciones con otras entidades se pueden observar en las etiquetas @OneToMany y @ManyToMany para las relaciones uno a muchos y muchos a muchos respectivamente. El parámetro *mappedBy* indica qué atributo de la otra entidad vamos a usar como clave foránea para la relación, como se muestra en la ilustración 27

```java
@Entity
public class Comentario {

    public static int indiceID=0;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int ID;
    // ID de un comentario para un negocio
    @Temporal(javax.persistence.TemporalType.DATE)
    private Date fechaCreacion;
    private String texto;
    @ManyToOne
    private Usuario creador;
    @ManyToOne
    private Negocio objetivo;
    @ElementCollection(targetClass=String.class)
    private List<String> likes;  // Guarda los tokens de quien han dado like
```

Figura 27 Ejemplo de código de la entidad Comentario

4.3 Detalles sobre implementación en el front-end.

Tras detallar en el apartado anterior la implementación de varios fragmentos del back-end. En este apartado se comentarán detalles que se deben tener en cuenta así como ejemplos de código de elementos clave en el mantenimiento y expansión de las prestaciones del front-end.

4.3.1 Enrutamiento de vistas en el cliente

En este proyecto las vistas de angular no se cargan por separado sino que su código html se va refrescando sobre una única vista, para realizar este proceso se utiliza un router outlet, este se declara en el html como un elemento más, y en él se cargan el resto de las vistas según el enrutamiento que designemos, en el caso de este proyecto el router outlet se ha declarado en app.component.html.

```
<router-outlet ></router-outlet>
```

Para definir una nueva ruta simplemente se vinculan un *path* y un componente Angular, esto se debe realizar en el archivo app.module.ts

```
imports: [
  RouterModule.forRoot([
    {path: 'Registro', component :RegistroComponent},
    {path: 'Negocio', component: PerfilNegocioComponent},
    {path: 'Login', component: LoginComponent},
    {path: 'Perfil', component: PerfilUsuarioComponent},
    {path:  'Mis-Negocios', component: MisNegociosComponent},
    {path:  'Crear-Negocio', component: CrearNegocioComponent},
    {path:  'BuscarNegocios', component: BuscadorNegociosComponent}
  ]),
  ReactiveFormsModule,
  FormsModule,
  AgmCoreModule,
  BrowserModule,
  HttpClientModule,
  AppRoutingModule,
  AgmCoreModule.forRoot({
      apiKey: "AIzaSyBz5Z7_UtVTEH0FG1ItXKBYsUJkpCai50k"
    })
],
```

4.3.2 Control de acceso en el cliente mediante tokens

Para gestionar la seguridad y el control de acceso de los usuarios en el lado del cliente, se utiliza el token, cuya generación se detalla en el apartado 4.2.2. Este token se almacena en una cookie cuando un usuario realiza un login exitoso, después este token es utilizado en las diferentes llamadas a la API cuando se intenta acceder a información exclusiva del usuario o cuando se intentan modificar los datos del mismo, así como al realizar comentarios y puntuar negocios.

Como podemos observar en las siguientes ilustraciones, en el constructor del servicio API del cliente se inyecta la dependencia del servicio de cookies que gestiona este token.

```
import { CookieService } from 'ngx-cookie-service';

@Injectable()

export class APIserviceService {
private apiKey: string;
private token: string;
private urlBase: string;

  constructor(
    private httpClient: HttpClient,
    private cookieService: CookieService,
  ) {
  this.apiKey = "AIzaSyDNNkuIfn-Ji2sjJs68BfRKA0fHUhxpxmA";
  this.token="";
  this.urlBase="http://localhost:8080/API/";
}
```

Figura 28 Declaración de la clase API del front-end

Una vez tenemos el objeto cookieService declarado podemos almacenar, borrar y modificar datos con facilidad, y al estar registradas las cookies en el navegador diferentes clases pueden utilizar el servicio sin necesidad de comunicarse directamente entre ellos. Para realizar las siguientes acciones tenemos los metodos set y delete con los que estableceremos el valor de la cookie, asignando a un valor clave, como se observa en la 3ª línea de código de la siguiente ilustración, donde se asigna a la cookie “UsuarioActivo” el token del usuario, aunque esta información puede ser desde una simple linea de texto a un objeto en formato JSON.

```
setUsuario(usuario: Usuario){
  this.usuarioActivo=usuario;
  this.cookieService.set("UsuarioActivo_id",""+usuario.id);
  this.cookieService.set("UsuarioActivo",usuario.token);
}
```

Figura 29 Ejemplo de inicio de sesión en el cliente

```
cerrarSesion():void{
  this.isloggedin=false;
  this.cookieService.delete("UsuarioActivo");
}
```

Figura 30 Ejemplo de cierre de sesión en el cliente

4.4 Despliegue en la nube

Una vez se tiene una versión de la aplicación lista para su despliegue en la nube se deben realizar los siguientes pasos.

El primero es asegurarse que en los directorios raíz de cada proyecto (front-end y back-end) existe un archivo Dockerfile, el cual nos sirve para indicar a Docker que hacer con nuestro proyecto una vez lo transforme en una imagen. Para este proyecto se han creado los siguientes archivos, el primero para el back-end y el segundo para el front-end.

Dockerfile

```
1  FROM openjdk:8
2  ARG JAR_FILE=target/*.jar
3  COPY ${JAR_FILE} basic-0.0.1-SNAPSHOT.jar
4  WORKDIR /opt/app
5  CMD ["java","-jar","/basic-0.0.1-SNAPSHOT.jar"]
```

Figura 31 Archivo Dockerfile para el back-end

```
# Stage 1

FROM node:10-alpine as build-step

RUN mkdir -p /app

COPY package.json /app

WORKDIR /app

RUN npm install

COPY . /app

RUN npm run build --prod


#Stage 2

FROM nginx:1.17.1-alpine


COPY --from=build-step /app/dist/prueba-angular /usr/share/nginx/html
```

Figura 32 Archivo Dockerfile para el front-end

Una vez tenemos nuestros ficheros Dockerfile, el proceso es idéntico tanto para el proyecto de Angular como el de Spring por lo que para el ejemplo se va a utilizar solamente la imagen de Spring. Para empezar abrimos una consola de comandos en el directorio raíz de nuestro proyecto y ejecutamos el comando *Docker build* , el usar -t nos permite asignar un nombre o "tag" a nuestra imagen docker, en este caso este tag sería tfg-backend.

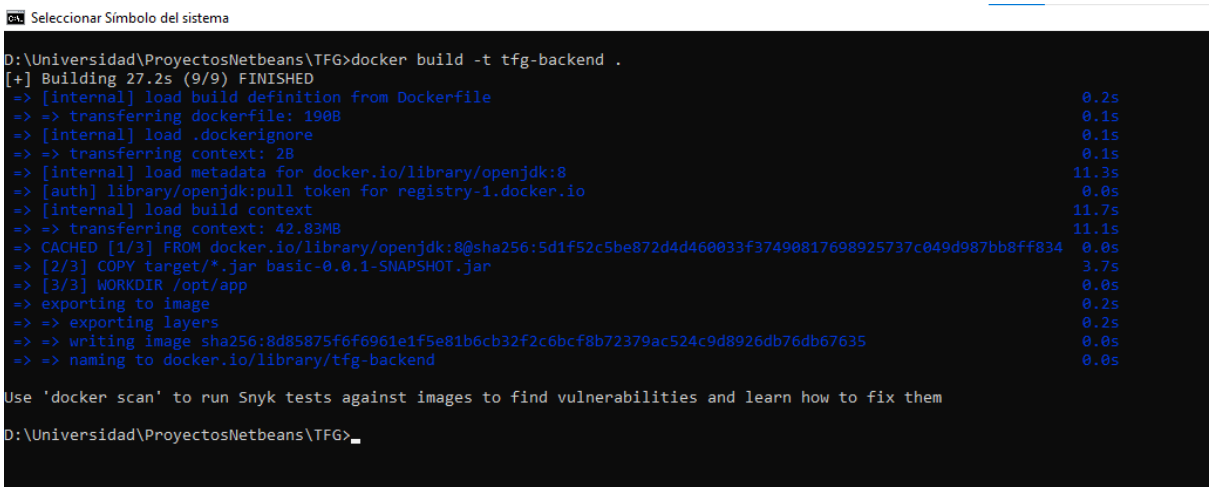


Figura 33 Ejecución comando Docker build

Tras tener nuestra imagen, para desplegar el servicio necesitamos una serie de requisitos previos, que son [12] :

- Tener una cuenta GCP
- Kuberctl, CLI tools para kubernetes
- Gcloud sdk, CLI tools para GCP

Una vez tenemos estos requisitos, y hemos creado previamente un proyecto en *Google Cloud* podemos subir nuestros contenedores a Google para desplegarlos. Para ello primero asignamos un nuevo tag a nuestra imagen previamente creada en el que indicaremos el servidor de Google al que subiremos la imagen así como el proyecto al que se subirá. Tras esto ejecutamos docker push para subir nuestra imagen

```
D:\Universidad\ProyectosNetbeans\TFG>docker images tfg-backend
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
tfg-backend   latest    8d85875f6f69   19 minutes ago   557MB

D:\Universidad\ProyectosNetbeans\TFG>docker tag 8d85875f6f69 eu.gcr.io/tfg-backend/api-backend

D:\Universidad\ProyectosNetbeans\TFG>docker push eu.gcr.io/tfg-backend/api-backend
Using default tag: latest
The push refers to repository [eu.gcr.io/tfg-backend/api-backend]
0b40af6be7ac: Pushed
083bd951d720: Pushing [==========>                                        ]  9.176MB/42.82MB
61ef3b3b77be: Layer already exists
e63cb31c0354: Layer already exists
c2e2307780ac: Layer already exists
7d890913ab69: Layer already exists
1235daf38153: Layer already exists
62d14713f2e9: Layer already exists
c2ddc1bc2645: Layer already exists
```

Figura 34 Ejemplo subida contenedor Docker a los servicios de Google

Una vez hemos subido nuestra imagen a Google entramos en nuestra cuenta de Google Cloud y en servicios seleccionamos Cloud Run

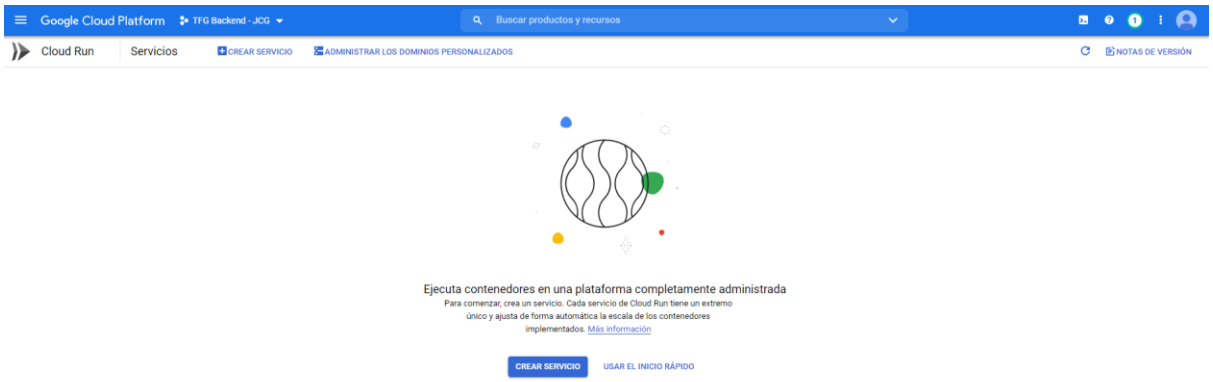


Figura 35 Vista previa Cloud Run

Tras seleccionar crear servicio damos un nombre a nuestro servicio recién creado y seleccionamos la región donde se alojará el servidor, en este caso se ha elegido Finlandia porque al ser un servidor con bajas emisiones es más económico.

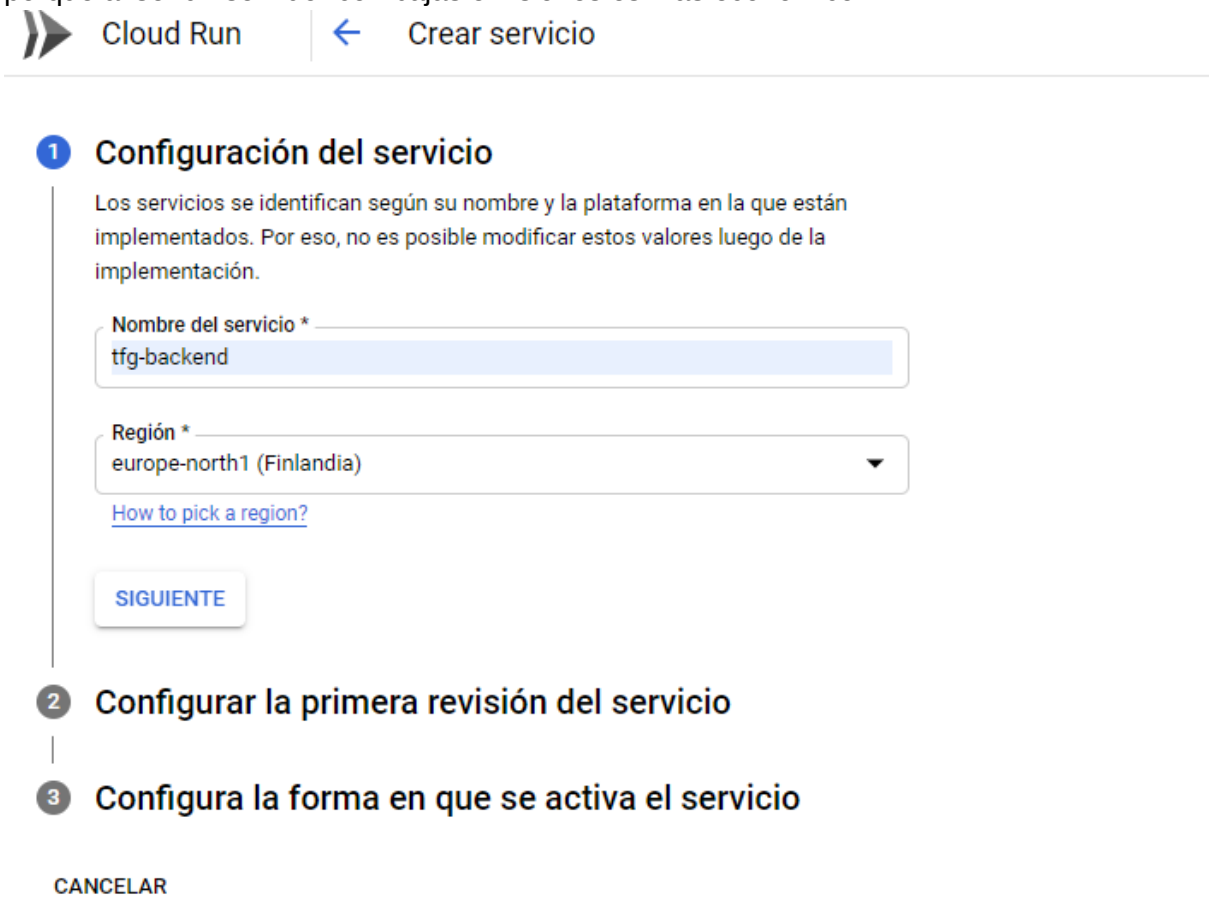


Figura 36 Creación de servicio Cloud Run

Tras esto, seleccionamos nuestra imagen entre aquellas que hemos subido previamente y en opciones elegimos permitir llamadas sin autenticar para crear una API pública.

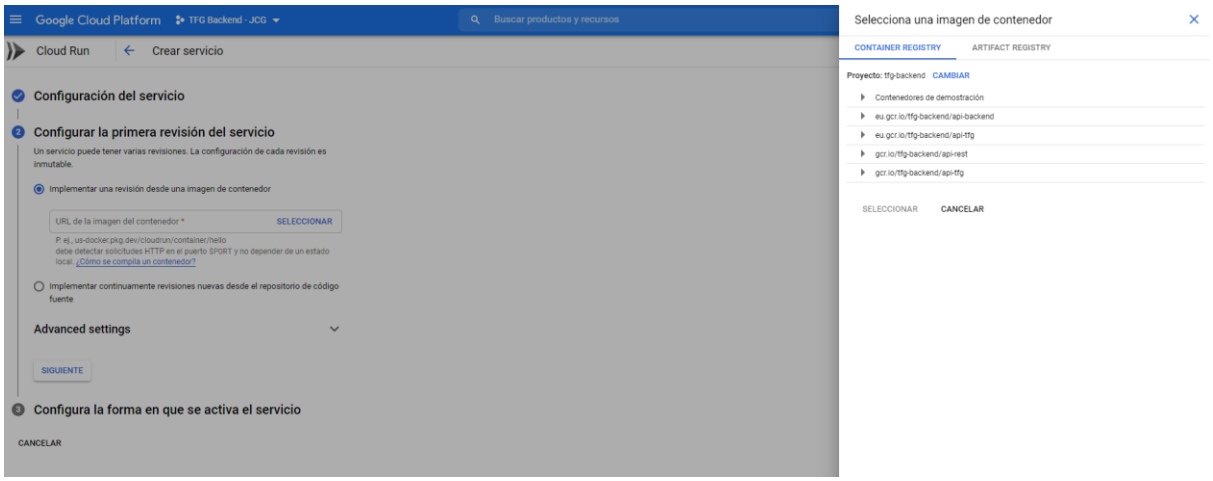


Figura 37 Selección imagen Cloud Run

Tras esto, ya tendremos nuestro servicio desplegado en los servidores de Google, los cuales nos proporcionarán una URL que podremos usar para acceder a ellos, se proporcionará una URL pública en los anexos para poder probar tanto la API como la aplicación front-end.

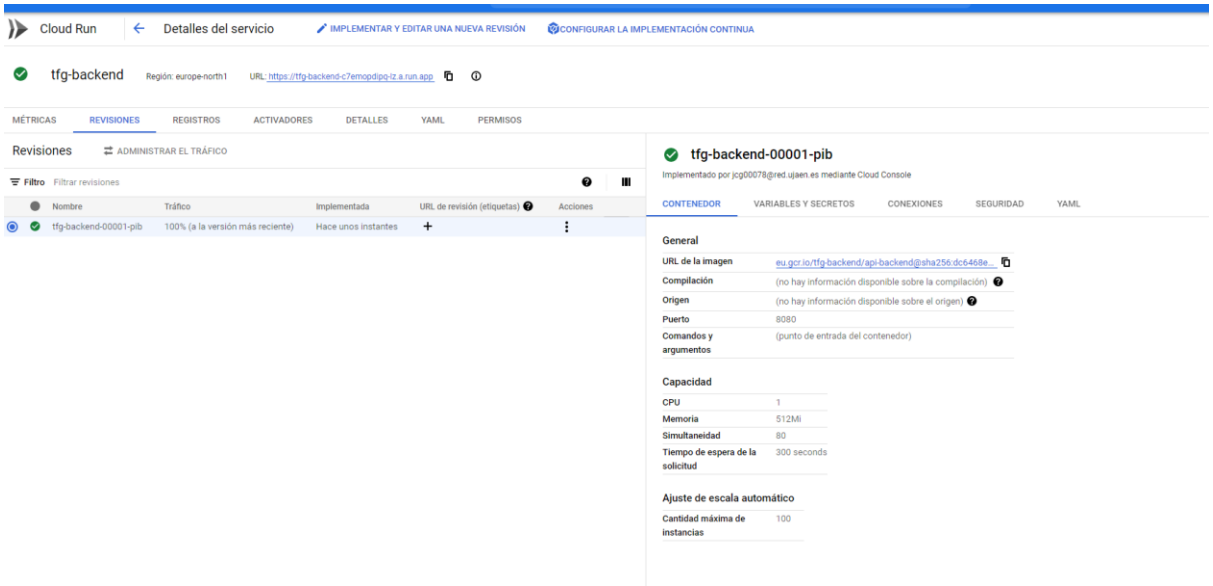


Figura 38 Servicio desplegado en Cloud Run

5 Pruebas

Para comprobar la integridad de las diferentes funcionalidades del sistema se han ejecutado las pruebas para validar los diferentes criterios de aceptación de las historias desarrolladas así como las pruebas en Postman para hacer funcionar correctamente todas las llamadas a la API. En estas pruebas se encontraron algunos errores que conllevaron modificaciones en la implementación. Para cada una de las 13 historias de usuario implementadas se realizaron pruebas en función de sus criterios de aceptación, para cada criterio se ejecutaron pruebas

positivas y negativas, es decir, si dados unos datos correctos el objetivo de la historia se cumplía y si en caso de proporcionar datos erróneos se devolvía un error o se notifica al usuario, dado que con estas historias abarcamos 16 criterios de aceptación diferentes se realizaron 32 pruebas de aceptación en total. Algunos ejemplos de los resultados más relevantes que arrojaron estas pruebas son los siguientes :

- Se encontró que un usuario podía puntuar varias veces el mismo lugar por un error de definición de las etiquetas JPA en la clase de Usuario, lo cual se corrigió.

- Se encontró que al ser el token generado por AES y contener caracteres extraños además de ser excesivamente largos, Angular encontraba algunos problemas al realizar comparaciones por lo que se modificó el código de generación de los tokens para reducir el tamaño de estos, lo que solucionó el problema.
- Al realizar pruebas con la biblioteca externa Places se descubrió que las búsquedas por tags de Google no funcionaban correctamente por un fallo en la implementación de la misma biblioteca, por lo que se modificaron los sistemas de búsqueda de tal forma que no filtrara directamente los negocios con la biblioteca externa sino que se ejecutarían búsquedas radiales, o por nombre y dirección y sobre los resultados de estas se realizaría un filtro con los tags usados en el sistema en vez de los propios de Google

Para ilustrar el proceso de pruebas en Postman se ha creado la siguiente tabla con los diferentes *end-points* y métodos, dado que en la API hay 15 combinaciones entre *end-points* y posibles operaciones que se pueden ejecutar en ellos, se han definido 4 pruebas, datos correctos, parámetros nulos, datos erróneos y datos conflictivos. En total se realizaron unas 60 pruebas sobre la API, la tabla a continuación es una muestra de los resultados más relevantes de estas pruebas.

End-Point	Método	Tipo de prueba	Respuesta	Comentario	Resultado
/Comentario	POST	Parámetros nulos	Bad Request	No se permitió crear el comentario con datos insuficientes	Prueba superada
/Usuario	PUT	Parámetros nulos	Bad Request	El atributo bio se había especificado como *required* cuando no era imprescindible, lo que causó una excepción	Prueba no superada
/Usuario/{id}/Seguir	POST	Datos Correctos	Not Found	El atributo id se había establecido como parámetro en vez de variable de ruta, por lo que el servicio de API no lo encontraba	Prueba no superada
/Comentario/{id}/like	POST	Datos Correctos	OK	Se permitía a un usuario puntuar positivamente más de una vez un mismo comentario, lo cual se corrigió	Prueba no superada
/Negocio/BusquedaRadio	POST	Datos Correctos	OK	El sistema devolvía negocios ya registrados, lo cual se corrigió	Prueba no superada
/Negocio	POST	Datos Correctos	OK	No se encontraron errores	Prueba superada
/Usuario	GET	Datos Correctos	OK	Se devolvieron los datos del Usuario correctamente, pero se eliminó el token de los datos que se devolvían por cuestiones de seguridad, el token solo se puede obtener a partir del login	Prueba superada
/Comentario	Post	Datos erróneos	Bad Request	No se permitió crear un comentario al proveer un id de negocio y un token de usuario no registrados	Prueba superada

6 Conclusiones

Tras la realización de este trabajo me gustaría comentar en primer lugar cómo he abordado los objetivos iniciales de este TFG. Para la compleción de estos objetivos iniciales se definieron una serie de historias de usuario que abarcaran los requisitos mínimos junto a otras para dar un mayor alcance al proyecto. Partiendo de los objetivos iniciales expuestos en el apartado 1.2 expongo las soluciones que se han aportado a cada uno de ellos, las cuales ya han sido desarrolladas e integradas en el proyecto.

Respecto al primer objetivo, se hizo un estudio del impacto y la importancia de las redes sociales en un contexto de un mundo globalizado y cada vez más conectado, donde las aplicaciones web y las redes sociales ocupan cada vez más tiempo y relevancia en nuestras vidas tanto a nivel social, económico e incluso político. A su vez se estudiaron y se presentaron diferentes soluciones con contextos similares a los planteados en este trabajo, como se refleja en los apartados 2.1 y 2.1.1.

Para el segundo objetivo, se decidió escoger una arquitectura cliente servidor basada en un api REST, con un enfoque MVC para el diseño de la aplicación, uno de los más utilizados en desarrollos web. La metodología de trabajo seleccionada fue de tipo ágil con iteraciones basada en elementos de SCRUM, ampliamente utilizada en los desarrollos de aplicaciones web. Se escogió también un framework para el *back-end* que permitiera el desarrollo de una arquitectura orientada a microservicios, que permite una rápida ampliación de las funcionalidades de los sistemas, utilizando una API REST para proveer información a los clientes webs. En el caso de el desarrollo front-end se escogió un framework como Angular, orientado a diseños web de tipo SPA, que mejoran la eficiencia y fue diseñado de forma modular, de forma que añadir nuevas vistas es tan simple como crear nuevos componentes de Angular

Por último, para el tercer objetivo se diseñaron e implementaron las historias de usuarios planteadas en la fase de planificación. Entre ellas cabe destacar las relacionadas con la búsqueda de negocios en la zona como la historia de usuario Filtrar negocio (CH18) y Localizar negocios (CH19). No obstante, algunas quedaron planificadas y no se pudieron desarrollar como se comentará en el siguiente apartado.

A continuación me gustaría comentar brevemente los detalles y las impresiones del trabajo que me han parecido más interesantes y que considero que me han servido para profundizar y ampliar mis conocimientos obtenidos durante la realización del grado. Lo primero que me

gustaría comentar es el uso de nuevas tecnologías en el desarrollo del TFG que no había utilizado en la carrera, como por ejemplo Angular o los servicios de despliegue en la nube de Google. Angular es un framework en expansión, cada día más utilizado en el ámbito empresarial por pequeñas y grandes empresas y creo que haber tenido la oportunidad de trabajar con el mismo me ha servido para crecer a nivel profesional. A su vez me gustaría recalcar que no solo el uso de nuevas tecnologías me ha parecido enriquecedor, sino que también el uso de otros frameworks como Spring y lenguajes como Java, que he usado a lo largo de mi grado en varias asignaturas, me han servido para profundizar en mi conocimiento sobre ambos a la vez que he podido adquirir experiencia de primera mano sobre sus ventajas y posibles inconvenientes.

También me ha parecido interesante la posibilidad de poder aplicar de forma práctica los conocimientos adquiridos a lo largo de la carrera en un proyecto personal. Esto me ha permitido obtener una mejor visión de conjunto sobre las cosas que he aprendido en estos años ya que, al dar un paso atrás y ver el desarrollo realizado en su conjunto en este trabajo, aprecio mucho más claramente como todo lo aprendido en el grado de una forma u otra sirven a un propósito. Es decir, a modo de analogía, no se trata solo ver cada uno de los engranajes de un motor por separado sino ver cómo el trabajo conjunto de cada uno de ellos permite la creación de algo mucho mayor. Desde la selección de la metodología de trabajo a las tecnologías seleccionadas así como el enfoque de arquitectura, elementos que durante la carrera se ven de forma separada, todos adquieren una importancia mucho mayor al ver como trabajan en conjunto para facilitar el desarrollo de una aplicación funcional partiendo de una simple serie de objetivos básicos.

6.1 Mejoras y trabajos futuros

Aunque se han cumplido los objetivos iniciales previstos para este TFG, algunas de las historias de usuario planteadas no han podido implementarse por los cambios de planificación que fueron necesarios. Aunque cabe destacar que con la arquitectura modular y distribuida planteada la integración de estas nuevas historias no requeriría demasiado esfuerzo y podrían ser desarrolladas, sin excesivas dificultades, con un poco más de tiempo Los cambios en la planificación antes mencionados fueron ocasionados debido a mi falta de experiencia en el uso de las nuevas tecnologías que decidí usar en este trabajo, por lo que no todas las historias planteadas pudieron ser incorporadas en la aplicación final. Las historias restantes por desarrollar planteadas en la planificación de este proyecto suman un gran valor a la aplicación final y serían el primer paso a desarrollar si se quisiera mantener y ampliar este proyecto Estas historias de usuario son: personalizar información negocio (CH09), suscribirse a un negocio (CH11), importar fotos de negocios (CH14), lista de favoritos (CH16), notificaciones de negocios (CH17), cambiar foto de perfil (CH20). Haciendo una estimación inicial todas ellas podrían ser acometidas en 2 - 3 iteraciones adicionales, aunque algunas como CH11 tengan una dificultad adicional al incluir la integración con sistemas de pago.

Otro tipo de mejoras podría ser la realización de integraciones con servicios como Google y Facebook usando por ejemplo sistemas para vincular cuentas, pudiendo iniciar sesión en nuestra red a partir de una cuenta de Google o mejoras en la interfaz. También se podrían

sustituir los mapas de Google Maps que se muestran en las vistas de los negocios por otros que solo incluyan los negocios registrados en el sistema.
.

Bibliografía

[1] «Market Snapshot: Desktop and mobile internet usage in 2020 | Telemedia ONLINE», nov. 09, 2020. [En línea]. Disponible en: https://www.telemediaonline.co.uk/market-snapshot-desktop-and-mobile-internet-usage-in-2020/https://github.com/skd1993/google-places-api-java#download

[2] «World Wide Web Technology Surveys». [En línea]. Disponible en: https://w3techs.com

[3] W. Morris, «8 Best PHP Frameworks for Web Developers», may 05, 2021. [En línea]. Disponible en: https://www.hostinger.com/tutorials/best-php-framework

[4] K. Chandrakant, «Why Choose Spring as Your Java Framework?», *Diciembre 13 2019*. [En línea]. Disponible en: https://www.baeldung.com/spring-why-to-choose

[5] K. Jemina, «Secrets of a nimble giant | The Guardian», jun. 17, 2009. [En línea]. Disponible en: https://www.theguardian.com/technology/2009/jun/17/google-sergey-brin

[6] R. Panko, «The Popularity of Google Maps: Trends in Navigation Apps in 2018 | The Manifest», jul. 10, 2018. [En línea]. Disponible en: https://themanifest.com/mobile-apps/popularity-google-maps-trends-navigation-apps-2018

[7] «Docker (software)», *Wikipedia*. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Docker_(software)

[8] M. Rehkopf, «Historias de usuario con ejemplos y plantilla». [En línea]. Disponible en: https://www.atlassian.com/es/agile/project-management/user-stories

[9] J. McIntyre, «MoSCoW or Kano Models – how do you prioritize?», oct. 20, 2016. [En línea]. Disponible en: https://www.hotpmo.com/management-models/moscow-kano-prioritize/

[10] «Bibliografía de Google para la API de GooglePlaces».

[11] «Biblioteca externa Google Places». [En línea]. Disponible en: https://github.com/skd1993/google-places-api-java#download

[12] F. Leiva, «Dockerizando y desplegando tu aplicación java». Disponible en: https://leiva.io/2019/04/16/dockerizando-aplicacion-java-para-desplegar-la-en-la-nube/

[13] «Salario medio analista España ». Disponible en:

https://es.indeed.com/career/analista-programador/salaries

[14] «Guía de instalación de Angular». Disponible en: https://angular.io/guide/setup-local

[15] C. Lasa Gómez, A. Álvarez García, R. de las Heras del Dedo, «Métodos Ágiles Scrum, Kanban, Lean», Anaya Multimedia, 2018

Apéndice I. Manual de Instalación del sistema

Para esta sección se distinguirán 2 apartados, el despliegue e instalación del front-end y el despliegue e instalación del back-end, estas instrucciones estarán referidas a un despliegue en local de la aplicación, ya que el despliegue online de ambas partes de la aplicación en los servidores de Google se desarrolla ya en el apartado 4.3.

Para el despliegue del back-end en un entorno local es imprescindible tener instalado, al menos la versión de Java, Java 8, así como sería recomendable pero no imprescindible el tener instalada la herramienta de desarrollo NetBeans, su despliegue en local es tan simple como importar nuestro proyecto y ejecutarlo, otra alternativa es ejecutar el fichero basic-0.0.1-SNAPSHOT.jar.

Para el despliegue del front-end, debemos instalar Angular CLI para ello ejecutaremos en la consola [14]

Figura 39 Instrucción instalación Angular

Una vez tengamos Angular instalado, entraremos en la carpeta de nuestro proyecto y ejecutaremos la orden ng serve –open, la cual desplegará nuestra aplicación en localhost:4200

Figura 40 Instrucción despliegue proyecto Angular

Por último se disponen de 2 cuentas de servidores de BBDD online que se pueden utilizar, sus datos son:

- Cuenta 1:
  Host: sql11.freesqldatabase.com
  Database name: sql11432274
  Database user: sql11432274
  Database password: KD3S47kkt9
  Port number: 3306
- Cuenta 2:
  Host: sql11.freesqldatabase.com
  Database name: sql11432275
  Database user: sql11432275
  Database password: UvWLFgiybe
  Port number: 3306

Apéndice II Manual de usuario

Crear cuenta

Para registrarnos en el sistema tenemos que acceder al formulario de registro que se encuentra en la barra superior

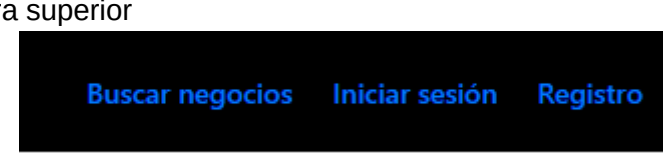


Figura 41 Barra superior proyecto Angular

Una vez entramos en el formulario solo tenemos que introducir nuestro nombre de usuario con el que otros usuarios nos identificarán y un correo electrónico y una contraseña para iniciar sesión, tras esto pulsamos el botón de registro y si nuestros datos son correctos se nos redirigirá a la página de login

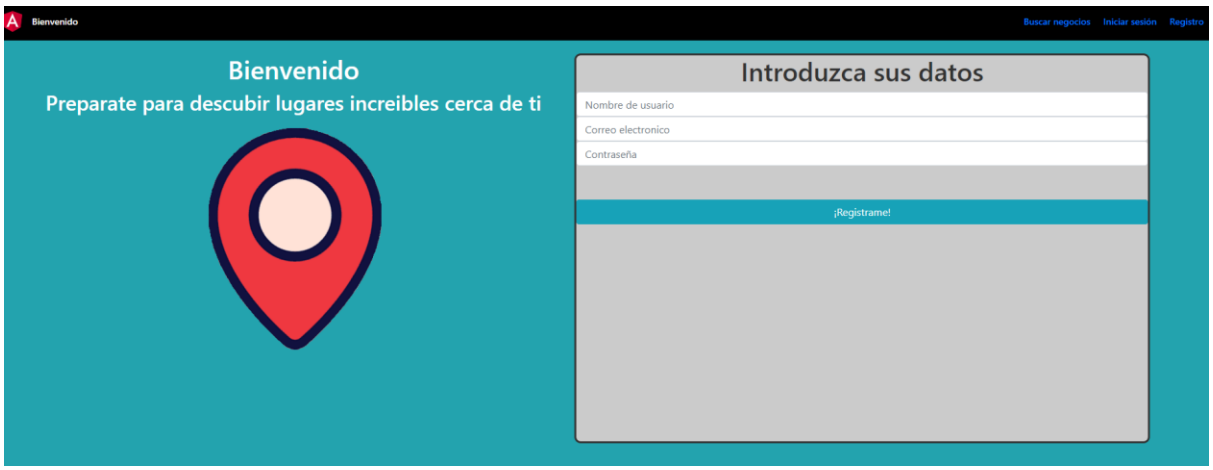


Figura 42 Vista de registro

Iniciar sesión

Para llegar a la vista de inicio de sesión podemos llegar desde el registro tras crear una cuenta satisfactoriamente o clicando directamente en la barra superior

Figura 43 Vista de login

Una vez introducimos nuestro correo electrónico y contraseña se nos redirige a nuestro perfil y la barra superior cambia mostrando nuevas opciones de navegación.

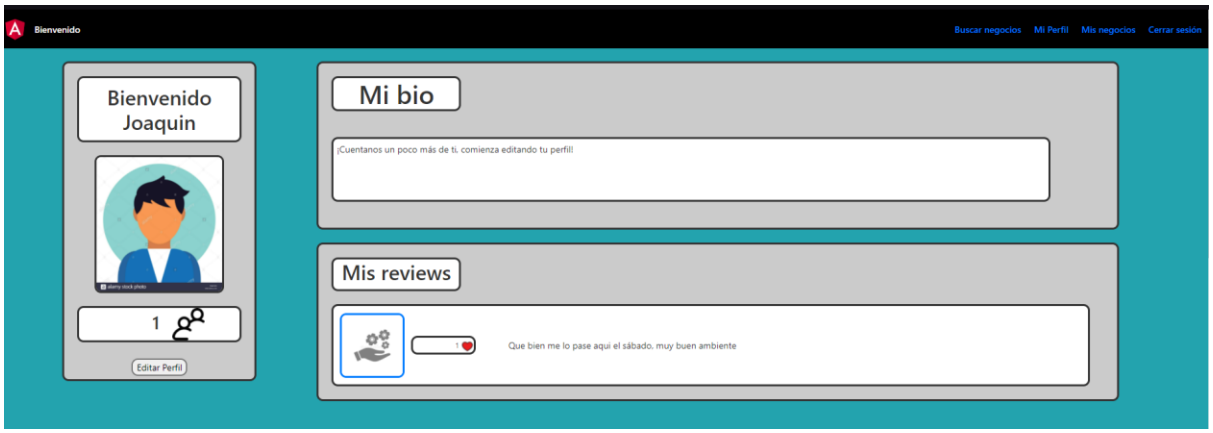


Figura 44 Vista de perfil de usuario

Para cerrar nuestra sesión clicamos en el enlace de cerrar sesión y seremos redirigidos a la página inicial y la barra superior volverá a su estado original.

Editar el perfil de usuario

Para editar la información básica de usuario clicamos el botón editar usuario en nuestro perfil, lo que cambia la vista al modo edición.

Figura 45 Botón de editar perfil

Figura 46 Modo edición perfil usuario

De este modo podemos modificar nuestro nombre de usuario y/o nuestra bio. Una vez hemos escrito los nuevos datos clicamos en confirmar cambios para guardar las modificaciones o en cancelar edición para volver a la vista normal de nuestro perfil.

Crear un negocio

Para crear un negocio debemos clicar primero en Mis negocios en la barra superior lo que abrirá la siguiente vista.

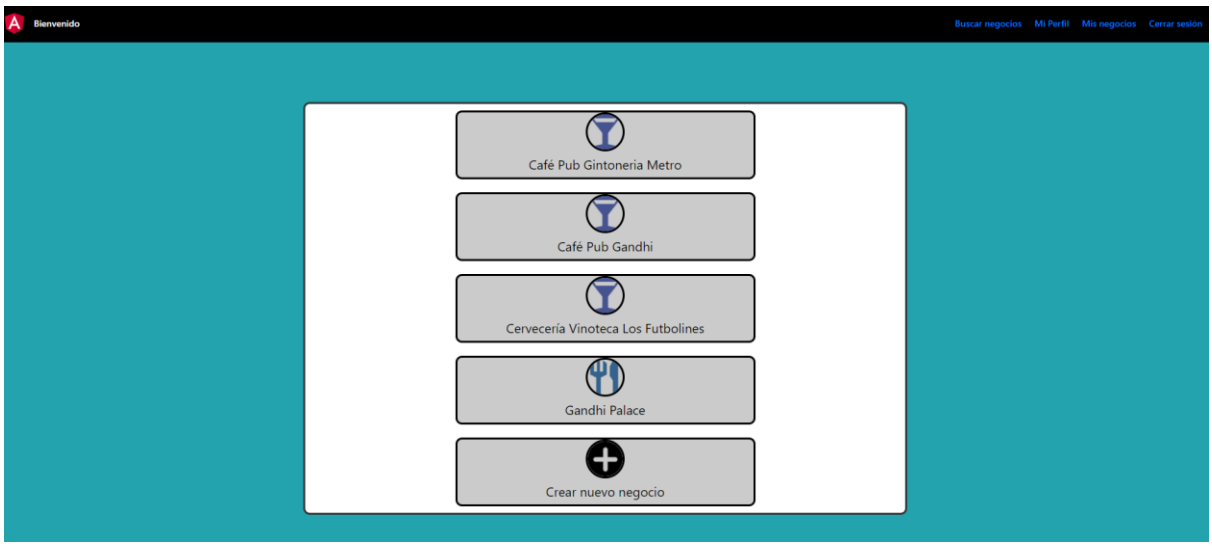


Figura 47 Vista crear negocio

En ella tendremos una lista de los negocios que tenemos registrados, para crear un nuevo negocio clicaremos en la opción de Crear nuevo negocio. Esto nos llevará a la siguiente vista en la que introduciremos el nombre de nuestro negocio y/o su dirección para localizarlo con los servicios de Google y clicaremos en importar mi negocio.

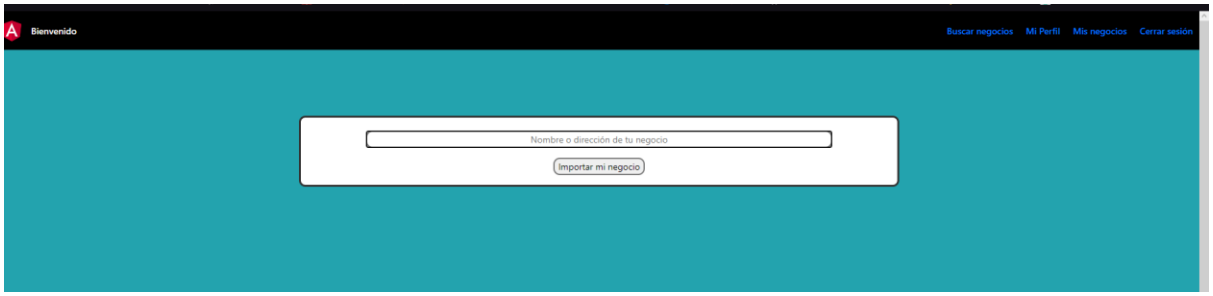


Figura 48 Vista crear negocio - importando tu negocio

Tras una breve espera se nos mostrará una lista de negocios entre los cuales seleccionaremos el nuestro.

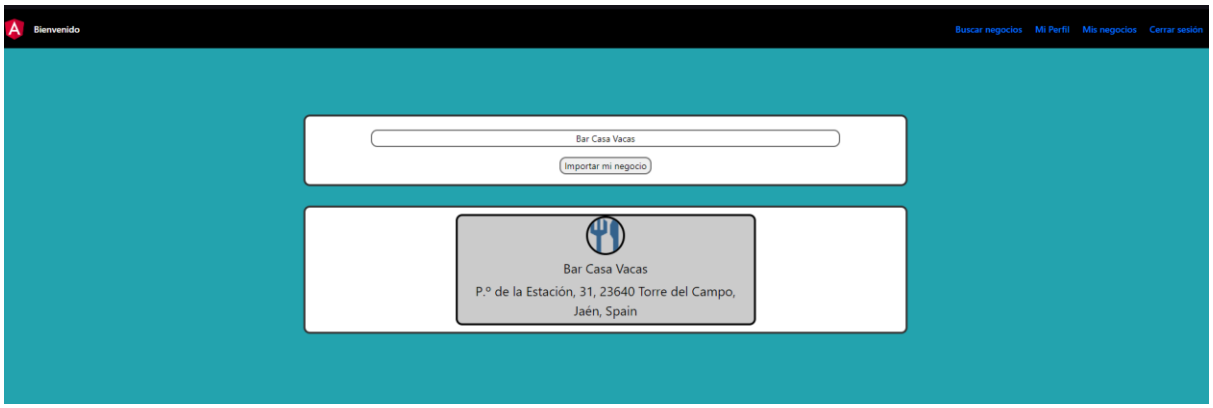


Figura 49 Vista crear negocio - resultado búsqueda

Tras clicar el negocio se registra automáticamente en el sistema y podemos accederlo desde la vista de Mis negocios.

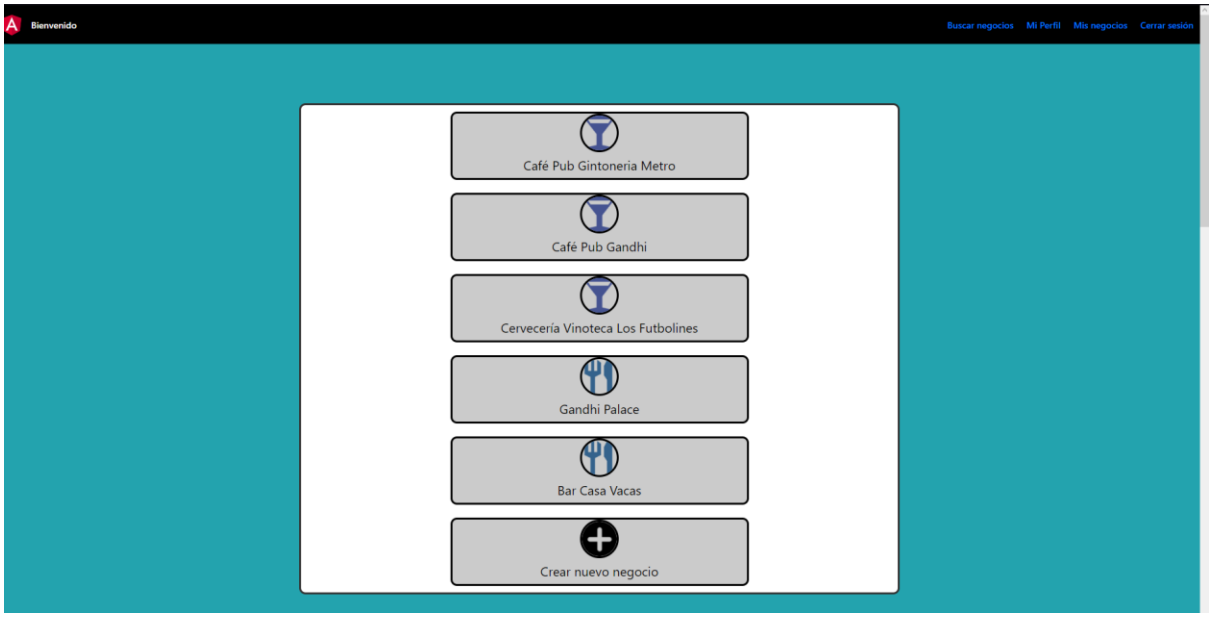


Figura 50 Vista mis negocios

Si hacemos click en cualquiera de estos negocios se nos redirigirá automáticamente a su perfil.

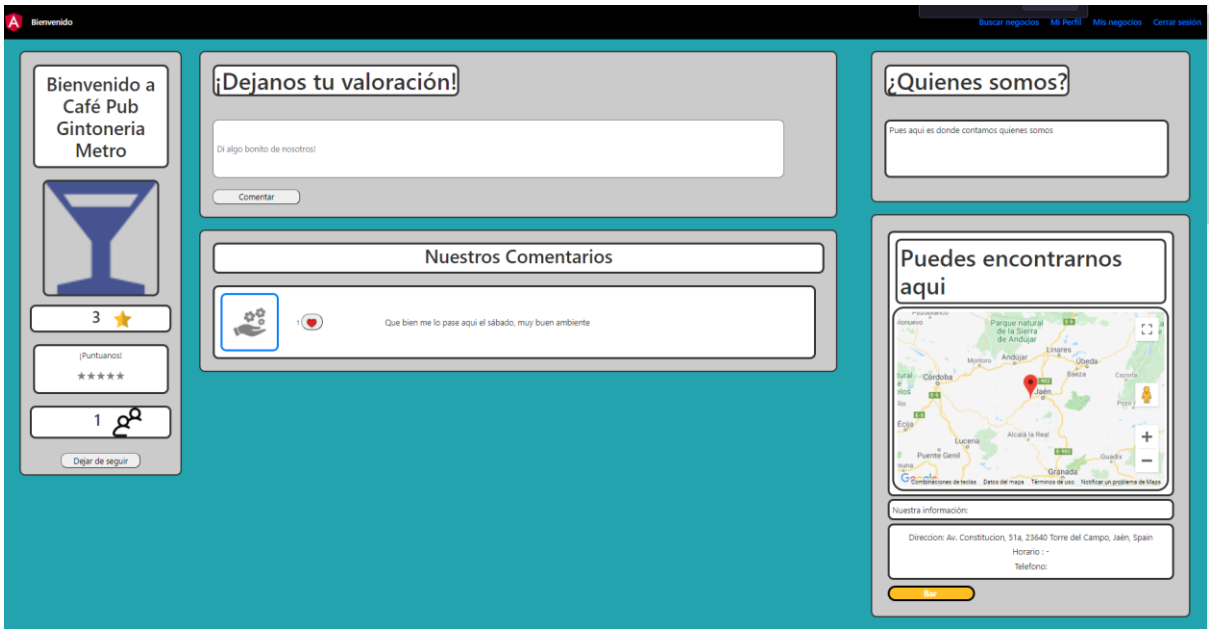
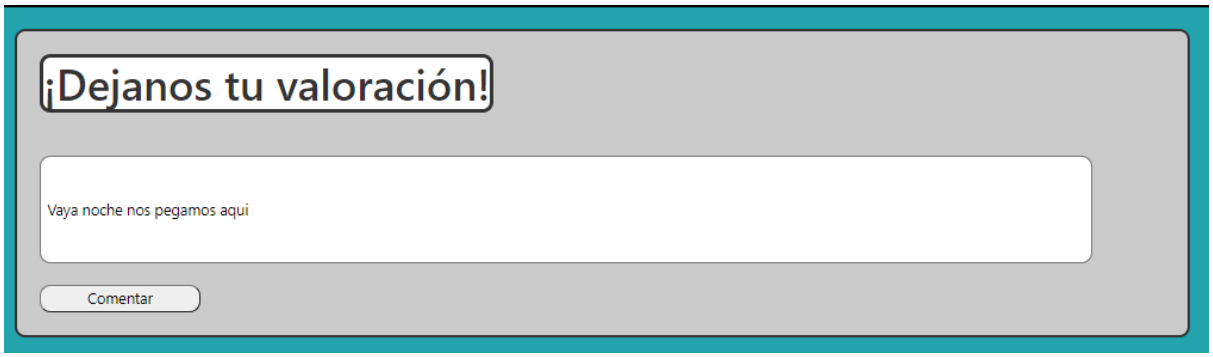


Figura 51 Vista perfil negocio

Crear y dar me gusta a un comentario

Una vez estamos en el perfil de un negocio podemos dejar un comentario escribiendo en la caja de la parte superior y pulsando el botón comentar.



Los comentarios creados para un negocio aparecerán justo debajo en su sección. Para dar me gusta clicamos en el corazón. Este corazón aparecerá de color rojo si ya hemos dado me gusta a un comentario, para retirar un like clicamos otra vez el corazón.

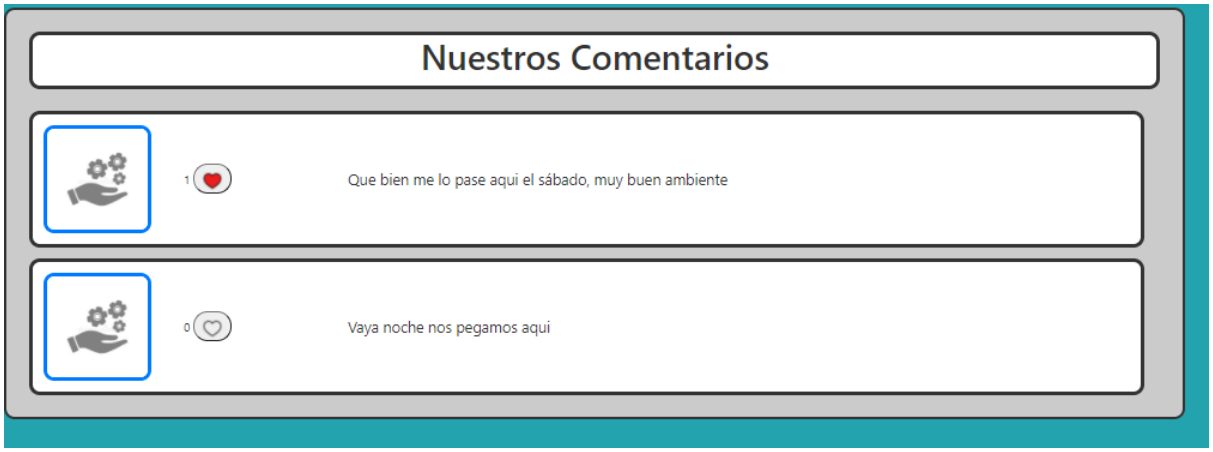


Figura 52 Vista perfil negocio - me gusta dado

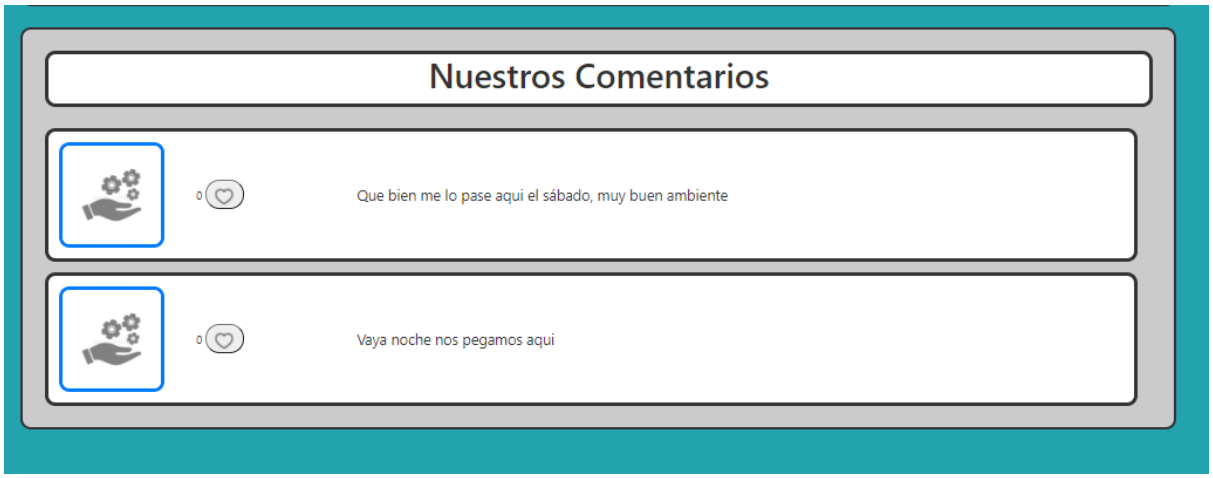


Figura 53 Vista perfil negocio - me gusta quitado

A su vez podremos clicar en las fotos de perfil de los comentarios para acceder al perfil de los usuarios que los postearon, si lo realizamos desde la vista del negocio o a los negocios si lo hacemos desde la vista del usuario.

Buscar un negocio

Para buscar un negocio clicamos en la opción de buscar negocios en la barra superior, lo que nos lleva a la vista de buscar negocio.

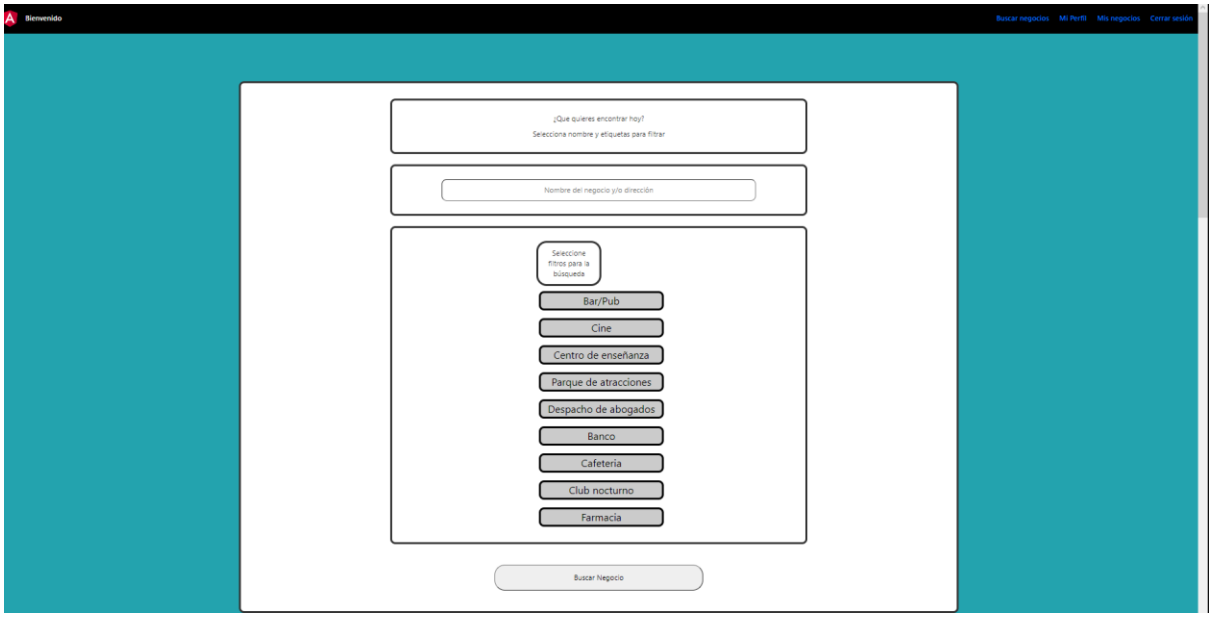


Figura 54 Vista buscar negocio

En esta vista podemos seleccionar los criterios de nuestra búsqueda filtrando por nombre del negocio y por etiquetas. Como se muestra en la imagen a continuación, la etiqueta seleccionada se resaltará en color verde. Tras seleccionar los criterios de búsqueda pulsamos el botón buscar.

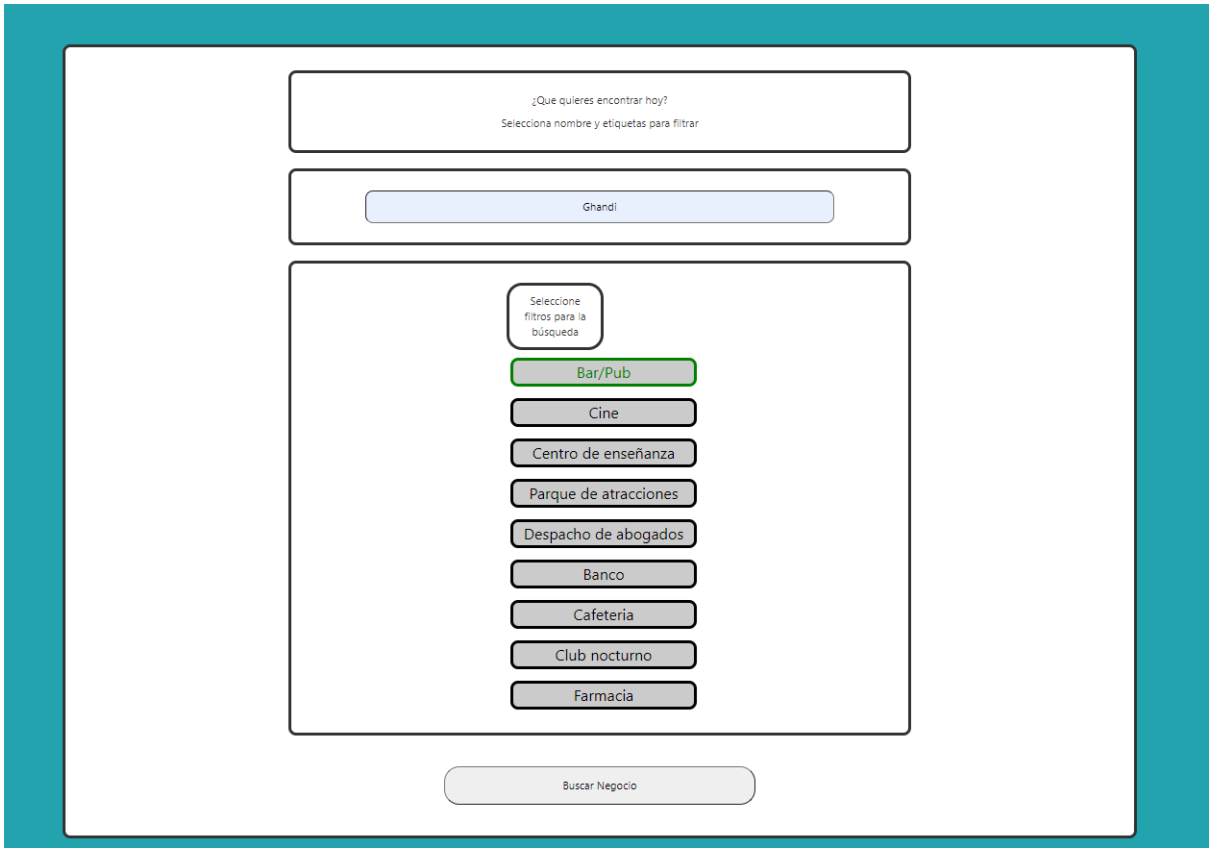


Figura 55 Vista buscar negocio - filtro aplicado

Un pequeño texto indicativo aparecerá sobre el botón de buscar para indicar que se está realizando la búsqueda.

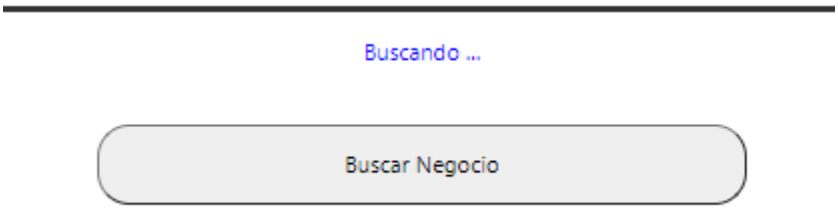


Figura 56 Vista buscar negocio - buscando

Tras la búsqueda los resultados se mostrarán en una lista desde la que podremos acceder al perfil de cada negocio o clicar en Volver a buscar para realizar otra búsqueda

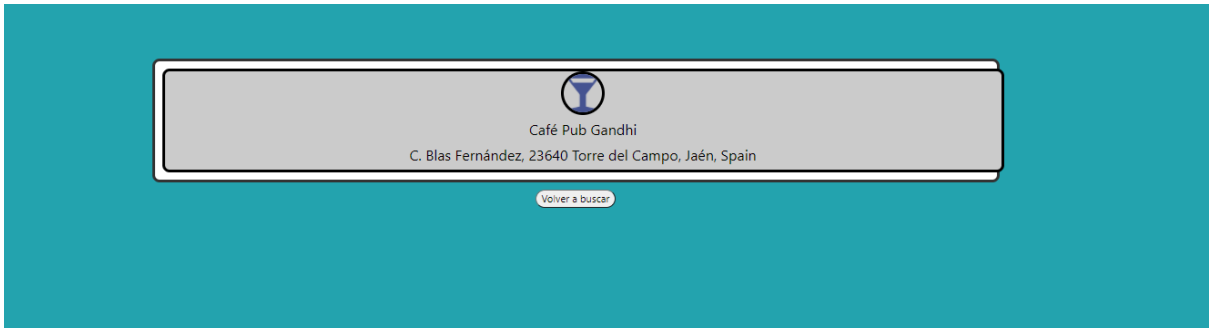


Figura 57 Vista buscar negocio - resultado búsqueda

Apéndice III Descripción de contenidos suministrados

Junto a la esta memoria se adjuntan los siguientes contenidos adicionales clasificados en las siguientes carpetas, las cuales se incluirán en un único fichero .zip comprimido:

- Angular: En ella se encuentran todos los archivos del proyecto de Angular para el front-end
- VisualParadigm: En esta carpeta se encuentran los archivos de VisualParadigm usados para documentar los diferentes diagramas que aparecen en la documentación.
- Documentación API: En esta carpeta se encuentra el fichero API.yaml que contiene la documentación de la API del back-end.
- Documentación extra: En esta carpeta se encuentra un fichero draw.io con el diagrama de arquitectura del sistema.
- Java: En ella se encuentran los ficheros del proyecto NetBeans del back-end así como los ejecutables java del mismo.

Apéndice IV Acceso a instalación de pruebas

Para poder probar el sistema desarrollado se ha habilitado un despliegue online en los servidores de Google que estará funcionando hasta el día 30/09/2021. Este despliegue está dividido en 2 apartados para probar por separado: la API y el backend. A la hora de probar la aplicación se debe tener en cuenta que los tiempos de respuesta pueden ser algo elevados dado que el despliegue está configurado con las mínimas prestaciones a fin de eliminar costes:

- La API del backend es accesible desde esta ruta : https://tfg-backend-entrega-c7emopdipq-lz.a.run.app/API/
- El front-end es accesible desde esta ruta: https://frontend-tfg-c7emopdipq-lz.a.run.app

Para facilitar las pruebas sobre el front-end se proveen una serie de instrucciones con los datos de ejemplo que se han introducido en la base de datos.

Se ha creado un usuario de prueba con datos ya modificados para hacer login, el correo a introducir es JoseTFGPrueba@gmail.com y la contraseña: 1234, a su vez si realizamos una búsqueda introduciendo Gandhi como nombre de negocio podremos encontrar algunos negocios de ejemplo sobre los cuales se han realizado comentarios.