Escuela Politécnica Superior

BlinDates

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Universitat d'Alacant
Universidad de Alicante

Autor:
Carlos Morales Navarro

Tutor/es:
José Vicente Berná Martínez

Junio 2018

Resumen

BlinDates es una aplicación móvil para la generación de citas a ciegas entre los usuarios registrados. Se basa en la compatibilidad calculada a través de un formulario que se realizará al comienzo del uso de la aplicación.

Para la localización de las citas, añadiremos un tercer agente en la aplicación, los restaurantes, que se suscribirán en la aplicación con el fin de registrar las reservas para las citas generadas y que sean ellos mismos los que administren el espacio y el menú ofrecido para cada día, a través de una plataforma web.

Motivación, justificación y objetivo general

Sumergidos en el mundo de las tecnologías, cada día se hace más importante la presencia en redes sociales para aspectos cotidianos de la vida. La amistad, el conocimiento o la opinión quedan diariamente reflejados y compartidos entre los usuarios de estas redes ampliando así el marco de la vida personal (ya no tan personal) que se ha arraigado fuertemente en la sociedad.

Existen muchas opciones actualmente dónde se nos facilitan estas tareas. Facebook, Twitter o Instagram son claros ejemplos de redes sociales que cuentan con una masiva actividad diaria. Facebook registra actualmente más de 2.167 millones de usuarios activos cada mes, con una media de uso de 21 minutos al día por usuario. Tanto es el alcance de esta red que en el año 2012 adquiere otra de las grandes potencias en este ámbito, Instagram.

Ya podemos imaginar que Instagram sigue los pasos de este gigante, no con tanto éxito, pero nada menospreciable: cuenta actualmente con más de 800 millones de usuarios activos al mes con un alcance que supera los 80 millones de fotos al día. Todas estas métricas, que nos ofrece Juan Carlos Mejía (Consultor en Marketing Digital y Social Media) [1], hablan por sí mismas, pero cabe recalcar que cada una de esas publicaciones que mencionamos significa una exposición de la vida personal y la actividad social de forma pública.

Además de las ya mencionadas, destinadas al uso más personal y social, se nos presentan otras tantas como Badoo, que cumplió los 300 millones de personas registradas el pasado 2016, o Zoosk, que le sigue los pasos con más de 27 millones de miembros en más de 80 países, cuya finalidad es la de poner en contacto a todos sus usuarios con un motivo en común: el amor. En estas redes sociales los usuarios interactúan entre sí con el fin de tratar de conocerse, y, si se da el caso, avanzar hacia una posible relación.

Profundizando levemente en los aspectos sociológicos de las personas y las redes sociales, nos encontramos en un punto en el que las personalidades reflejadas en las redes sociales son las caretas que esconden una verdad que no se conoce. Es decir, a través de estas redes conocemos de cada persona únicamente lo que esa persona quiere que conozcamos de ella. Además, cuando la variable "amor" entra en juego, queda virtualizada en una sociedad modernista donde ha decaído en apología sexual en la mayoría de los casos.

Cristina Miguel, profesora en la Facultad de Comunicación de la Universidad de Leeds [2], describe muy bien esta nueva realidad como: *"La tecnología no genera pautas sociales. La gente ya no cree en el 'para toda la vida' y busca cosas más prácticas, más a corto plazo, y ahí entran estas aplicaciones. Las aplicaciones y sites de ligue cuestionan varios mitos del amor romántico. El mito del azar, del carácter fortuito del encuentro de pareja; el de la media naranja, es decir, solo hay una persona en el mundo a la que estamos destinados y a la inversa; y el del amor ciego y no calculado, ahora sustituido por la elección razonable e interesada".*

Así es la tendencia actual en el amor, un tema que existe desde el comienzo de la humanidad. Bien es cierto que el amor siempre se encuentra en constante evolución, adaptándose a los aspectos y las creencias de las diferentes culturas dónde podemos encontrarlo. Aunque no tenemos que volver la vista atrás muchos años para darnos cuenta que el cambio ha sido trascendental, pasando del "para siempre" al "para un rato". Pero bueno, no estamos aquí para realizar una crítica sociológica sobre la situación del amor en la era en la que nos encontramos.

Pocos somos los románticos que todavía buscamos ese cosquilleo, ese nerviosismo que genera una primera cita, esa incógnita del futuro, ese anhelo de un todo y el miedo a nada. Sin embargo, la cadena de televisión Cuatro, con el programa First Dates [3], ha conseguido recrear en cierta manera todo eso que citábamos, adaptado a la época actual. Esas citas a ciegas, antes organizadas por amigos/as que no querían vernos solos, o que más bien necesitaban una ayuda para que su cita fuese bien, y nos emparejaban con el amigo/la amiga de su pareja; ahora son posibles gracias a la cadena de televisión donde, después de pasar el casting oportuno, buscarán esa persona con la que creen que tendremos mayor compatibilidad para compartir una cena en su plató, junto con otras tantas personas que harán lo mismo. Es una lástima que detrás de muchas de ellas se encuentre un carácter de publicidad y marketing con el fin de generar audiencia, comprensible después de todo.

Pero es justamente este programa, el que nos hace pensar que todavía, en la era tecnológica, existe un hueco donde la gente puede arriesgarse por amor, donde un programa puede darte la oportunidad de conocer eso que al final todos buscamos por naturaleza. Y siendo la era de la tecnología, ¿por qué no utilizar la propia tecnología para cubrir ese espacio del romanticismo?

Así nace la idea de BlinDates (del inglés, cita a ciegas).

BlinDates es una aplicación que permite generar citas a ciegas entre los usuarios registrados basados en la compatibilidad a través de un formulario de registro que se realizará al comienzo de su uso. Basándose en prioridades como distancia, preferencia sexual o edad, calculará a través de la información sobre cada usuario, quién es la persona más compatible.

De cara al modelo de negocio, permitiremos que los restaurantes se suscriban en la aplicación con el fin de registrar las reservas para las citas generadas y que sean los propios restaurantes los que administren el espacio y el menú ofrecido para cada día.

Fuera del marco televisivo pierde potencia frente a First Dates: la incertidumbre es un arma de doble filo. Si bien es cierto que el no conocer a la otra persona, no saber a quién vas a encontrar, puede generar una inseguridad, lo enfrentamos con un potente seguimiento y verificación de registros cotejando la información con canales como Facebook o Google. Así mismo, la cita se realiza en un sitio púbico y con conocimiento del establecimiento. Y por otro lado, estamos en un momento donde lo nuevo y lo desconocido es atractivo, sumado al contante uso de las tecnologías donde el mundo de las aplicaciones está creciendo de forma exponencial. Son oportunidades para descubrir y experimentar, y quién sabe, quizá crear tendencia.

Agradecimientos

Sin duda quiero agradecer a José Vicente Berná todo el esfuerzo realizado para proporcionar las herramientas adecuadas para completar este proyecto. Los materiales y consejos aportados a lo largo del desarrollo han sido un gran apoyo.

No puedo dejar fuera a mi familia que desde el día uno fue un gran apoyo y me ha animado y conducido a conseguir lo que me proponía, ¡aunque me haya hecho de rogar!

Quería mencionar también de forma especial a Jorge López Arangua, porque cada momento de la carrera sin duda se lo debo a él. Ha sido largo el camino recorrido juntos y muchos los momentos en los que una mano amiga me ha ayudado a superar los baches.

Por último, agradecer a una persona especial para mí, Claudia Sánchez González, que ha sido la que ha peleado casi con más fuerza que yo para que pueda cerrar esta etapa. El TFG, desde su principio a su conclusión es algo que le debo a ella, aportando en ideas, revisiones y, sobre todo, siendo mi apoyo cuando los días eran más complicados.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

La evolución de la tecnología ha revolucionado hasta un mundo tan abstracto como es el amor. Con infinidad de redes sociales donde conocer gente, en apenas unos años hemos sustituido miles de escenas románticas por mensajes o publicaciones online. No cabe duda de que con la evolución de la sociedad, estas prácticas se han vuelto más cotidianas que aquellas a las que estábamos acostumbrados anteriormente. Las relaciones han pasado a ser de ámbito público.

Entre toda la inmensidad de redes sociales que existen actualmente, encontramos una amplia gama especializada en la temática del amor. En ellas existe un objetivo claro, exhibir públicamente los perfiles de los usuarios con fines amorosos. Según nos explica José Antonio Molina de Peral [4], psicólogo: "La tecnología ha conllevado un cambio significativo a la hora de establecer relaciones, de comunicarse. La gente recurre a una serie de redes en las cuales se evita el cara a cara, se ahorra el cortejo tradicional. Es fácil e inmediato. Puedes estar en casa, sin vestirte, sin arreglar... y sin embargo estás ligando, a través de Internet, evitando por ejemplo la vergüenza de acercarte a alguien en un bar. Podríamos decir que te hace el trabajo sucio".

El amor ha cambiado mucho en los últimos años. La manera de sentirlo, de expresarlo, de vivirlo y de encontrarlo es totalmente distinta. Todo ello ha dado lugar a un nuevo escenario en el que las opciones se multiplican prácticamente de manera exponencial y en el que si hace unos años la mejor arma de seducción con la que podías contar era una bonita sonrisa acompañada de receptividad al 100%, hoy, como no tengas un Smartphone o un ordenador y conexión a internet, difícilmente podrás encontrar introducirte de lleno en el mundo del amor; o al menos esa parece ser la tendencia de nuestros días. Pero, dentro de todo ese cambio, ¿no existe hueco para los románticos que todavía buscamos sentir la pasión por el amor?

BlinDates es una aplicación que busca cubrir ese hueco, donde los románticos puedan subsistir en el mundo tecnológico. Este proyecto consiste en la creación de una plataforma especializada en citas a ciegas. Con este servicio pretendemos que la tecnología ayude a cubrir esos encuentros que antaño existían, que aceleraban el corazón, que te hacía preguntarte si esa noche conocerías al amor de tu vida. Así mismo, pretende ser únicamente un medio de concretar la cita, eliminando la posibilidad de conocer por medios digitales a la otra persona.

El objetivo es crear una plataforma donde ofreceremos a los usuarios la experiencia de las citas a ciegas. Completando un pequeño cuestionario de tipo personal, un usuario pueda generar citas a ciegas en una fecha y un lugar concreto, realizando el sistema un cálculo de compatibilidad con otros usuarios. Será necesaria la existencia de un tercer agente para poder cubrir la necesidad completamente: los restaurantes. Al incluirlos dentro de la ecuación, seremos capaces de ofrecer al usuario el lugar exacto de la cita, gestionando la reserva de forma automática, y proporcionando así un lugar que evoque el ambiente que citas de esta índole requieren.

Figura 1. Esquema visual del uso de BlinDates
(Fuente propia)

Haremos entonces foco en los siguientes aspectos:

- Los restaurantes podrán darse de alta en BlinDates con el fin de exhibir su local a los usuarios y ser elegidos como anfitriones para las citas generadas.
- Gestionarán dentro del sistema las reservas y menús disponibles para la elección de los detalles de la cita.
- Los usuarios podrán darse de alta en BlinDates con el fin de vivir la experiencia de las citas a ciegas.
- Completarán dentro del sistema la información necesaria para encontrar a la persona compatible para la cita, pudiendo elegir la fecha y el lugar de ésta de entre las posibilidades que el sistema ofrezca.

Siendo un proyecto ambicioso y complejo, ya que supone la definición de varios sistemas para interactuar con diferentes usuarios, la reducción de tiempo y coste de implementación se hace indispensable debido a la limitación de tiempo que tenemos para el desarrollo de éste (300 horas, según lo establecido por las especificaciones del TFG).

De cara a evitar que los restaurantes busquen un lucro excesivo a través de nuestra aplicación, tendrán predefinidos una serie de precios para que así los usuarios, al mismo tiempo que generan la cita, puedan saber cuánto van a gastar en el menú. Además, dado que BlinDates busca hacer de cada cita algo especial, dónde efectivamente el amor sea el trasfondo de todo encuentro, los restaurantes se responsabilizarán de crear el ambiente propicio para ello.

Además, podemos poner una pequeña carga de ingresos en el lado de usuario, con cuentas “Freemium” en el momento del registro y ampliables a un estado “Premium”. Pero nunca utilizaremos la reducción de funcionalidades con el fin de que los usuarios se suscriban.

BlinDates no es una aplicación como las que actualmente se encuentran en el mercado, donde se busca la conexión con personas de forma masificada para aumentar las posibilidades de éxito. BlinDates busca hacer algo especial, distinto, dónde los usuarios que se dispongan a la cita lo hagan de su mejor forma posible. Busca recuperar ese nerviosismo antes de conocer a esa persona, que, quién sabe, quizá sea la que tanto había esperado.

2. Estudio de viabilidad

BlinDates es un proyecto ambicioso. El proyecto combina elementos tanto de carácter tecnológico como externos, ampliando la lista de objetivos a conseguir.

Si contemplamos el ciclo de vida completo del proyecto, definiremos tres fases: desarrollo, captación de restaurantes y publicidad, marketing y captación de usuarios; teniendo en cuenta que en cada una de las fases serán diferentes los riesgos que debamos asumir.

Para analizar cada uno de los elementos que encontramos en las diferentes fases, hemos realizado un diagrama Lean Canvas [5], el cual nos propone una estructura donde, por un lado, tenemos el mercado, la parte más complicada de gestionar, y, por otro lado, tenemos nuestra empresa, su entorno, procesos y sus activos.

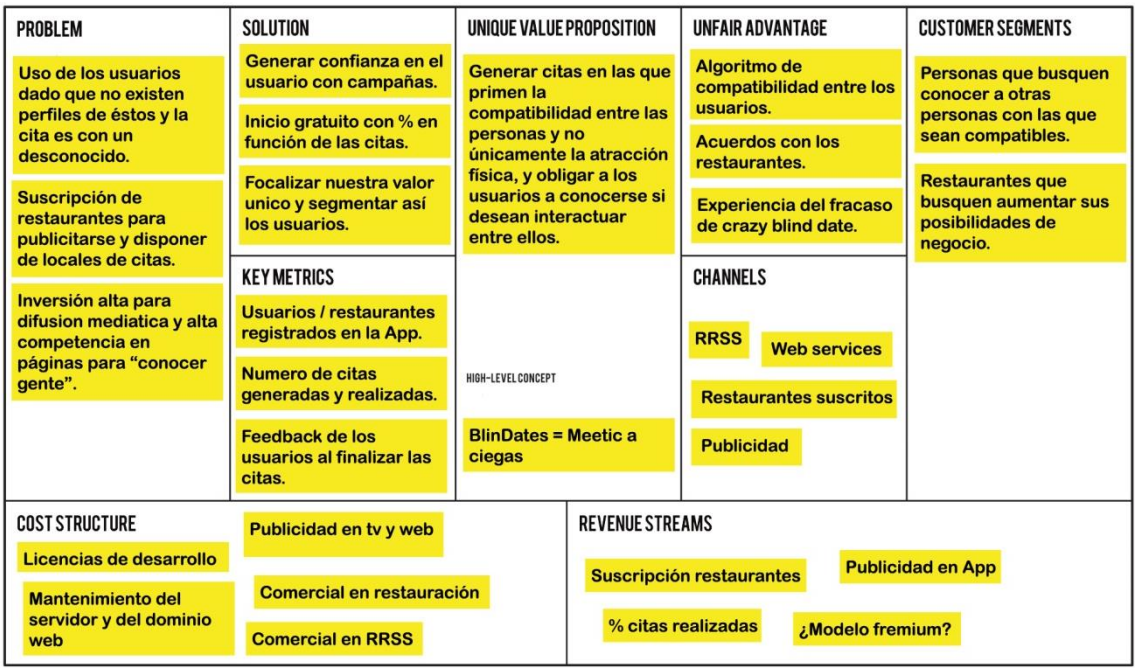

Figura 2. Tabla Lead Canvan para proyecto BlinDates.
(Fuente propia)

Como reflejaremos más adelante en el estado del arte, el mercado al que nos enfrentamos es un mercado al alza, con muchos competidores, pero, a su vez, muchas posibilidades. Otro punto a favor es que nuestro perfil de usuario potencial está acostumbrado a que las plataformas incluyan unas pasarelas de pago asociadas al uso del portal. En nuestro caso nos planteamos un posible ingreso a partir de un modelo *Freemium*, donde no limitaremos la aplicación en cuanto a funcionalidades, si no en cuanto a número de citas activas de forma simultánea, debiendo abonar una suscripción para disfrutar de la aplicación sin límites.

Nos enfrentamos entonces a asumir el coste de la estructura, el cuál computaremos en un alto porcentaje haciendo las labores de desarrollador y comercial, quedando así un coste derivado de la estructura necesaria (servidor, dominio, licencias…) y el coste de publicidad y marketing para la puesta en marcha del mercado.

2.1. Análisis de riesgos

BlinDates es un proyecto tecnológico donde tendremos que suplir todas las diferentes fases que en ellos se incluyen. No únicamente será en el desarrollo de la aplicación, donde asumiremos el aprendizaje de nuevos lenguajes de programación. Será necesario también el aprendizaje de creación y gestión de servidores, mantenimiento de dominios y gestión de DNS, sistemas de despliegue… Este conjunto será uno de los grandes retos del inicio del proyecto. Además, la situación personal del momento incluye un horario laboral que tendrá un fuerte impacto en el tiempo a poder dedicar al desarrollo del proyecto.

Basándonos entonces en las buenas prácticas de la metodología SCRUM, y añadiendo en la ecuación los riegos antes mencionados, estableceremos una contingencia en los elementos a desarrollar de un 20% del tiempo estimado.

3. Estado del arte.

La búsqueda de pareja por Internet es un negocio al alza, la gente trabaja mucho y su tiempo para encontrar parejas se reduce. Ligar en Internet es una realidad que cada vez está más aceptada en la sociedad. Es un hecho. Así nos lo explica Daniel Aparicio en su artículo sobre ligar por internet [6].

Virginia Collera nos cuenta en su reportaje sobre ligar en tiempos modernos [7] cómo el ritmo acelerado de vida y la tecnología están cambiando el arte de la seducción. Nadie tiene tiempo de nada, y la ley de la oferta y la demanda dicta las normas en el mercado del ligue. Internet se ha convertido en el escaparate donde nos exhibimos y volcamos nuestra vida privada. El análisis y la gestión de esos datos que cedemos libremente nos pueden ayudar a buscar lo que queramos, ya sea un compañero para toda la vida o una relación pasajera.

Las webs para ligar, encontrar pareja o tener encuentros esporádicos con mayor acogida en España tienen como denominador común un diseño atractivo, práctico y sencillo, además de servicios similares, como buscador, test de personalidad (unos más exhaustivos que otros) y chat (que suele ser de pago), pero cada una de ellas intenta dar su "toque personal". La mayoría de estas páginas webs cuentan además con una app con la que podrás permanecer en conexión desde tu Smartphone.

Los tiempos han cambiado, de la misma manera en la cual ya no concebimos no usar el móvil para comunicarnos, cada vez concebimos menos no usarlo para seducir: Carmen Jané nos presenta un estudio sobre las estadísticas de uso de aplicaciones de esta índole [8], citando a Jordi Bernabeu, que afirma que “las redes sociales o las apps son un espacio más de relación”.

Antes cuando queríamos conocer a gente nueva para encontrar pareja salíamos por la noche con nuestros amigos, ahora simplemente bajamos la mirada a nuestro móvil y comprobamos cómo va nuestro perfil en nuestra aplicación para ligar preferida.

Hay que reconocer que, ahora, con tanta aplicación, lo tenemos mucho más fácil - aunque sólo sea llegar a la deseada primera cita - puesto que las aplicaciones están fragmentadas y cada una de ellas tiene un público muy concreto con unos objetivos claros que ayudan a que sus usuarios consigan el tipo de relación que buscan. Veamos qué opciones tenemos:

3.1. Tinder

Probablemente una de las aplicaciones gratuitas para ligar más utilizadas de la actualidad. Tinder [9] es tan fácil y práctica que no hay mucho que explicar con relación a su mecanismo. Lo único que se deberá de hacer es deslizar a la izquierda el perfil del usuario que no le interese, y, a la derecha, los que lograron captar la atención. Si la persona a la que se le ha aceptado devuelve su interés, la plataforma automáticamente creará un chat para la conversación entre ambas personas. Permite conversar sin mostrar toda su información o perfil, ajustable en la plataforma.

A través de la geolocalización, esta aplicación ofrece la opción de buscar a usuarios que estén a unos cuantos kilómetros a la redonda; ya sean 5, 10, 20, 50, 100, etcétera. Entre algunas de sus novedades se encuentra la búsqueda de intereses musicales entre ambas personas, conectando los perfiles de usuario con la plataforma Spotify.

Figura 3. Captura de uso de la aplicación Tinder
(Fuente http://puamore.com/tinder/)

3.2. Happn

Si te has cruzado en la esquina de la calle con alguna persona y deseas devolver unos segundos para hablarle, con Happn [10] podrás hacerlo desde la comodidad de tu dispositivo móvil. Perfecto para aquellos que cruzan miradas a mitad de calle y sienten un pequeño *feeling*. Happn permite conocer el nombre y perfil del usuario que te has encontrado en cualquier sector de la ciudad. Visualiza el número de veces que se han cruzado, la hora y lugar exacto.

La aplicación mostrará el perfil del usuario que ha dado like a vuestras fotos, y, si hay un gusto mutuo, Happn establecerá un chat para iniciar la conversación. Además, nos permitirá también el registro en la aplicación a través de redes sociales externas como Facebook.

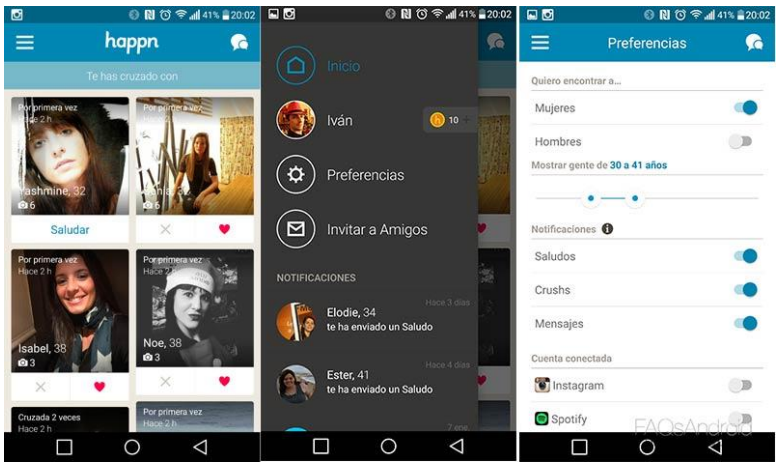

Figura 4. Captura de uso de la aplicación Happn
(Fuente https://www.happn.com/es/)

3.3. Badoo

Se podría decir que Badoo [11] es la aplicación para ligar más vieja de todos los tiempos. Los inicios de Badoo comenzaron en formato web. Después, al ver la gran cantidad de usuarios que se beneficiaban y encontraban pareja a través de la plataforma, se implementó su versión para dispositivos móviles.

Su uso es simple: aporta la información, gustos, rutinas y todo lo que sea importante en vuestra vida para conseguir usuarios con gustos similares. Mientras más información en relación con intereses y gustos personales se complete en el perfil, será más sencillo encontrar una pareja o amistad.

Permite puntuar fotografías y buscar afinidades mediante una especie de juego denominado "Encuentros". Ordenada, intuitiva y con multitud de opciones, es una buena herramienta para hacer amistades, encontrar a personas con intereses comunes o conseguir citas.

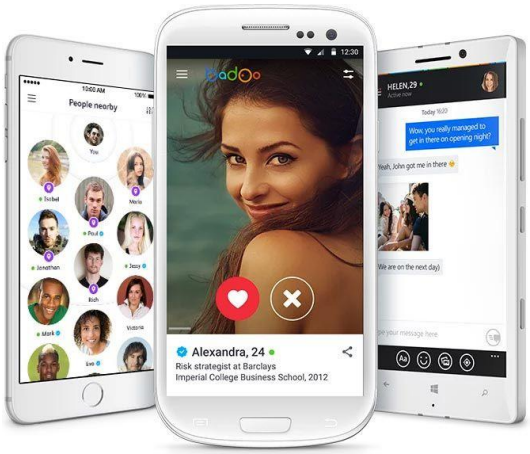

Figura 5. Captura de uso de la aplicación Badoo
(Fuente http://conocergenteweb.com/badoo-espana-opiniones-como-registrarse/)

3.4. Meetic

Meetic [12] cuenta con un potente buscador que nos permite encontrar personas afines a nosotros, en el que podemos discriminar con diferentes criterios como puede ser la edad, gustos, nacionalidad, estudios y muchos más. Además de darnos la opción de mostrarnos nuestras parejas ideales, cuenta con un servicio “turbo” que permite contactar de forma rápida con otras personas: tu perfil estará destacado durante un cierto tiempo, por lo que muchos usuarios podrán verte y así tener más posibilidades de éxito.

Como elemento extra, suele ofrecer un servicio de organización eventos y fiestas reales para que los solteros se puedan conocer entre ellos; sin duda un servicio acertado que permite romper la frialdad que supone la búsqueda de la pareja en la red.

Figura 6. Captura de uso de la aplicación Meetic
(Fuente https://iliusstda-a.akamaihd.net/)

3.5. Grindr

Grindr [13] ha cobrado una gran popularidad entre el público gay, dado que está diseñada especialmente para este colectivo y también para el de personas bisexuales. Es una app para buscar pareja gay que permite conocer hombres de nuestra ciudad con tan solo personalizar nuestro perfil, explicando un poco cómo somos y qué es lo que buscamos.

Aparecerá un mosaico de fotos ordenadas según su situación geográfica respecto a nosotros y a los que estén conectados. Para contactar con la persona que nos gusta solo hay que pulsar sobre su perfil e iniciar una conversación privada, compartir fotografías, etc., además podremos guardar las conversaciones si nos han resultado amenas.

Figura 7. Captura de uso de la aplicación Grindr
(Fuente https://www.tuexpertoapps.com)

3.6. Crazy Blind Date

Podemos concluir que actualmente existe un amplio mercado de aplicaciones móvil con finalidades amorosas, y con los datos de cada una de ellas determinamos que la demanda es amplia, y al alza.

Podemos determinar una serie de patrones de servicios y funcionalidades comunes para todas estas aplicaciones (perfiles, mensajería…), pero como vemos todas ellas se alejan de la idea principal en la que se basa nuestra aplicación: la cita a ciegas.

	Tinder	Happn	Badoo	Meetic	Grindr
Perfil	Muy reducido	Reducido	Completo	Completo	Muy reducido
Cuestionario	No	No	No	Si	No
Mensajería	Si	Si	Si	Si	Si
Match	Si	Si	Si	No	No
GPS	Si	Localización por cruce	Si	No	Si
Buscador	No	No	Si	Si	Si
Pago	No	No	Si	Si	No

Tabla 1. Tabla de funcionalidades en aplicaciones de citas
(Fuente propia)

Como hemos visto en el apartado anterior, hay una gran variedad de herramientas destinadas a la búsqueda de encuentros amorosos, pero hasta ahora no hemos podido ver ninguna que explote el concepto de cita a ciegas.

En una investigación más profunda encontramos únicamente una aplicación con características similares a la que aquí vamos a desarrollar ha visto la luz en el mercado, y ha fracasado: Crazy Blind Date [14].

El software es capaz de organizar una cita casi al instante. Los usuarios simplemente deciden las noches en las que quieren quedar con alguien y el horario y escogen un lugar favorito (un bar, un restaurante u otro local). La aplicación hace el resto. Cuando llega el día señalado entra en juego la monetización del servicio pues debes comprar la información del lugar para veros realmente, aportando a este usuario un poco de credibilidad extra, y sobre todo seguridad.

Crazy Blind Date comunica el sitio exacto del encuentro a las dos personas y les dice a ambas con quién van a quedar. La selección se realiza en base a sus gustos y compatibilidades. La gracia del asunto es que ninguno puede ver cómo es la otra persona, puesto que sus imágenes de perfil aparecen en forma de fractal desordenado, haciendo más difícil que se reconozcan en el momento de la cita. Después de la cita en cuestión es posible calificar cómo ha resultado la otra persona. Aquellos que tengan mayor puntuación pasan a ser ‘Kudos’ y obtienen un estatus superior. Para quedar con ellos y que el algoritmo los tenga en cuenta, hay que comprar créditos. El fracaso de esta aplicación se atribuye al hecho de la necesidad del pago para poder llevar a cabo la cita, sumada al desconocimiento del tipo y éxito de la cita.

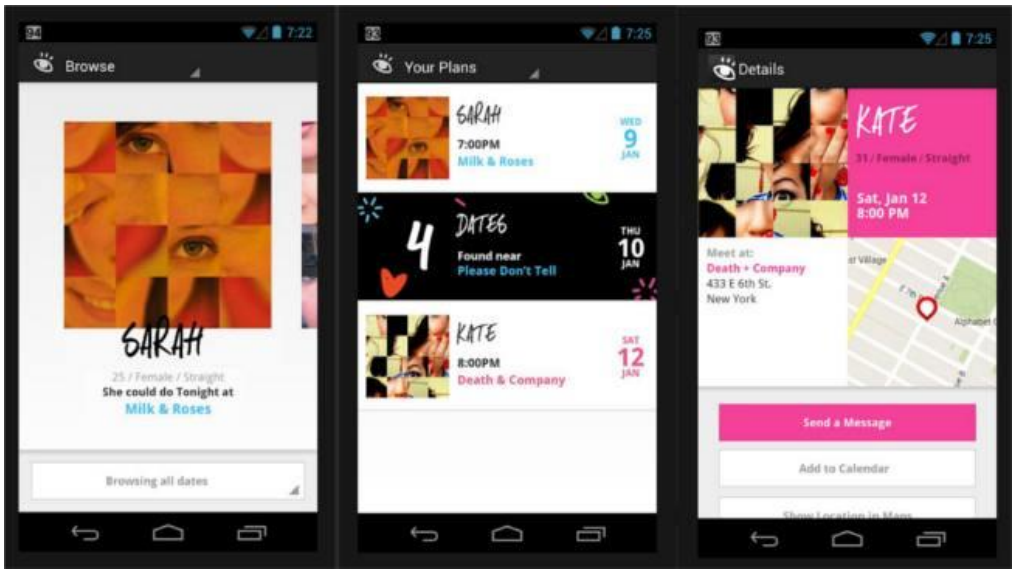

Figura 8. Captura de uso de la aplicación Crazy Blind Dates
(Fuente http://static4.businessinsider.com/)

La pregunta entonces es, si Crazy Blind Date ha fracasado, ¿por qué entonces apostar por una nueva aplicación de características similares?

Las distinciones de nuestra aplicación con las que actualmente se encuentran en mercado son claras: estas aplicaciones buscan la generación de perfiles para la interactuación de los usuarios dentro del sistema, con una finalidad monetaria basada en tipos de cuenta, funcionalidades… Tanto Crazy Blind Dates como nuestra aplicación buscan la interactuación de los usuarios fuera del sistema. Tratan de generar un encuentro entre los usuarios para eliminar esa capa de conexión y volver a las relaciones sociales cara a cara.

Pero Crazy Blind Date sigue balanceando la carga de financiación a la parte de los usuarios, donde son éstos los que, para poder llevar a cabo la cita, tienen que realizar los pagos correspondientes. Esto supone que el usuario está realizando el pago para poder "dejar de usar la aplicación" y dar pie al contacto cara a cara, lo que da lugar a un descenso del uso y de la finalidad de esta aplicación, lo que, en mi opinión, lo ha llevado al fracaso.

BlinDates, sin embargo, modifica el origen de su monetización cargándolo sobre los restaurantes que realizan las suscripciones en el sistema con el fin de que las citas se realicen en sus establecimientos. Además, son ellos los encargados de dar la personalidad y el enfoque de marketing dentro de nuestro sistema con el fin de ser atractivos de cara a los usuarios para que las citas se realicen en su establecimiento y no en otro.

La idea en BlinDates es combinar usuarios, plataforma y los establecimientos para las citas. Este tercer elemento es inexistente en el resto de plataformas. Con esto, BlinDates genera tráfico hacia los establecimientos, y es precisamente esto lo que genera la fuente de ingresos. Este además será otro de nuestros puntos de distinción frente a la ya mencionada aplicación Crazy Blind Date. En nuestro caso, ofrecemos a los usuarios un lugar de confianza para que la cita tenga lugar, con un compromiso de parte de los usuarios y del establecimiento para hacer de la cita algo especial.

3.7. Conclusiones

BlinDates es una apuesta contra un mercado al alza. Como vemos cada día es mayor el número de aplicaciones orientadas a la búsqueda de relaciones amorosas, y mayor es el consumo de éstas. Pero, aunque el trasfondo del objetivo sea el mismo, BlinDates no solo busca poner a sus usuarios en contacto para que puedan conocerse a través de nuestro sistema (a grosso modo, las aplicaciones analizadas no distan tanto de las redes sociales como Facebook, únicamente nos proporcionan un público más específico y, en algunos casos, algunos detalles más profundos de las personalidades. En otros incluso llega a ser más superficial). BlinDates busca volver a hacer especial una cita en un mundo dónde la tecnología ha apagado la chispa del amor.

Aplicaciones como Tinder o Happn apenas van más allá de lo físico. Proporcionan un sistema de interés causado por primeras impresiones.

Aplicaciones como Meetic profundizan un poco más en las características de sus usuarios para buscar compatibilidades mayores entre ellos a través de un test de personalidad, pero mantienen la necesidad de un sistema tecnológico dónde sus usuarios podrán conocerse (mientras pagan por hacerlo).

Y en eso se traduce la búsqueda del amor a día de hoy: usar la tecnología para buscar personas a las que poder conocer a través de estos sistemas. BlinDates busca romper con esta idea: ¿Por qué no utilizar la tecnología para "dejar de utilizarla"?

En BlinDates los usuarios no utilizarán la aplicación para conocer a personas. La aplicación es un sistema que les proporcionará la información del lugar y la fecha donde conocerá a una persona con la que tienen cierta compatibilidad. Al contrario que en resto de aplicaciones analizadas, nuestros usuarios no tendrán ningún tipo de impresión de su acompañante previa a la cita, lo que se traduce en la vuelta de esa chispa, de ese cosquilleo, de ese ¿cómo será él/ella? Además, el usuario tendrá que salir de casa, apartar la mirada del móvil y centrarse en conocer a quién tiene delante. Esta es claramente nuestra apuesta en un mundo cuya tendencia es la de la comunicación por medio de sistemas tecnológicos.

Uno de los valores añadidos que posee BlinDates es que además entra en juego un tercer agente, los restaurantes. En otras aplicaciones son los usuarios los que tienen que interactuar para conseguir la cita, y después plantearse el dónde, el cuándo… Con la aparición de los restaurantes en nuestra aplicación eliminamos esa incertidumbre y ese primer paso a la hora de planear la cita: nuestros restaurantes se encargarán de todo.

BlinDates busca que lo especial vuelva a ser especial.

4. Objetivos

El objetivo general de este trabajo es el de crear un sistema que permita a dos personas tener una cita a ciegas, utilizando esta idea para “redescubrir la chispa del amor”.

Para que esta plataforma sea accesible deberá cumplir los siguientes sub-objetivos:

- Facilitar su accesibilidad desde cualquier lugar y a cualquier hora para su interacción, y por tanto crear una aplicación móvil compatible con la multiplataforma.
- Establecer un mecanismo que permita el emparejamiento efectivo, para lo que se utilizarán formularios que recabarán información suficiente junto a algoritmos que formalicen la compatibilidad.
- Generar un entorno en el que brindemos a los usuarios la información y las herramientas necesarias para formalizar el lugar y la hora de la cita.
- Facilitar la aparición de un tercer agente, los restaurantes, con el fin de presentarlos como los lugares donde realizar las citas, creando una aplicación web para la gestión de su local dentro de nuestra aplicación.
- Crear un sistema de confianza para el usuario mediante la generación de formularios de valoración una vez la cita ha finalizado. Recogeremos información tanto de la cita en sí (si se ha realizado, grado de compatibilidad que has encontrado…) como de la atención recibida por parte del restaurante. Buscaremos:
    - Medir el grado de satisfacción de nuestros usuarios con el algoritmo que genera la compatibilidad entre los candidatos para las citas generadas.
    - Identificar los restaurantes que no cumplan con los mínimos establecidos en el momento de la suscripción en nuestra plataforma. Utilizaremos un sistema de *strikes* cuyo resultado final será el de la cancelación inmediata de la suscripción del restaurante.
- Presentar y acercar la idea de una forma confiable al público, entendible y amena, que despierte su interés, a través de una *landing page* (página web) que se incluirá dentro de un marco corporativo y será la utilizada para la difusión publicitaria.
- Analizar las diferentes posibilidades de monetización que se nos presentan para nuestro sistema y concluir el modelo a utilizar.

5. Metodología

Blindates es un proyecto con una base conocida, siendo ésta los primeros requisitos y especificaciones del sistema. Pero, al igual que cualquiera de las aplicaciones sociales, tenderá a la evolución en función de la petición del usuario. Para la planificación del desarrollo de Blindates, elegiremos por tanto un modelo de metodología que nos permita adaptarnos, sin un alto impacto, a los posibles cambios que surjan, ya sea durante el desarrollo previsto como su entrega, o más adelante, si el proyecto continúa evolucionando.

Con estas premisas, la metodología Scrum [15] nos permitirá definir los requisitos y las especificaciones del sistema hasta lograr la primera versión funcional del producto. A su vez, podremos ir identificando las posibles funcionalidades o mejoras que queramos introducir en nuestro sistema una vez superada esa primera versión y añadirlo a nuestro espacio de planificación de desarrollo.

En nuestro caso, el uso de esta metodología nos obligará a adoptar los diferentes (y principales) roles que la metodología define en una única persona. Nos permitirá crear una base de proyecto estable de cara a posibles incorporaciones futuras en el equipo, asumiendo cualquiera de los roles. No será por tanto una metodología Scrum por definición, sino una adaptación de la metodología para el desarrollo unipersonal, aplicando los mismos procesos que define la metodología.

Para establecer la metodología en nuestro proyecto, en primer lugar definiremos el proceso de cada ciclo de desarrollo. En Scrum, un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite. Así pues, basándonos en la planificación predefinida por nuestro tutor, establecemos un tiempo de diseño de la aplicación más el desarrollo de ésta en un tiempo máximo de tres meses. Dado el limitado tiempo de desarrollo del que disponemos, estableceremos los ciclos de desarrollo en dos semanas con el fin de obtener un mayor número de entregables y feedback de la evolución del producto.

Debemos, además, definir dentro de cada ciclo de desarrollo dónde tomarán parte cada uno de los roles principales de la metodología (especificación, desarrollo y *testing*). Al comienzo de cada ciclo deberá suceder la especificación precisa de cada una de las funcionalidades que vayan a ser implementadas, una posterior estimación de cada una de las tareas en las que hemos dividido el ciclo, para así confirmar qué elementos serán los que finalmente podamos llevar a cabo, y un proceso de testeo donde confirmar que los elementos desarrollados funcionan como se esperaba. Así, dadas las funcionalidades a asumir en cada ciclo y el tiempo del que disponemos (escaso, debido a factores externos, principalmente el trabajo), el esquema de cada ciclo de desarrollo será el siguiente:

Sprint	Semana	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
X	1	Análisis y especificaciones (Sprint X)				Estimaciones	Desarrollo	
	2	Testing (Spint X -1)				Desarrollo		

Tabla 2. Planificación del ciclo de desarrollo.
(Fuente propia)

Por último, nos queda definir la herramienta que utilizaremos para gestionar todos los elementos ya definidos. Team Foundation Server es un producto de Microsoft que nos proporciona un entorno donde gestionar todas las características de un proyecto tecnológico basado en una metodología Scrum. Nos proporciona además otras herramientas (control de versiones o gestión de despliegues del producto) que mencionaremos más adelante y que integraremos con otros apartados de nuestro proyecto.

Team Foundation Server es una herramienta web que nos permitirá gestionar los ciclos de desarrollo y las tareas previstas para cada uno de ellos. En primer lugar dispondremos de un espacio donde definir todos los posibles elementos de trabajo que vayamos identificando. En segundo lugar, dispondremos de diferentes espacios divididos para cada uno de los ciclos donde añadiremos al principio de cada ciclo los elementos en los que trabajaremos. Además, podremos definir los diferentes *bugs* que podamos identificar dentro del sistema.

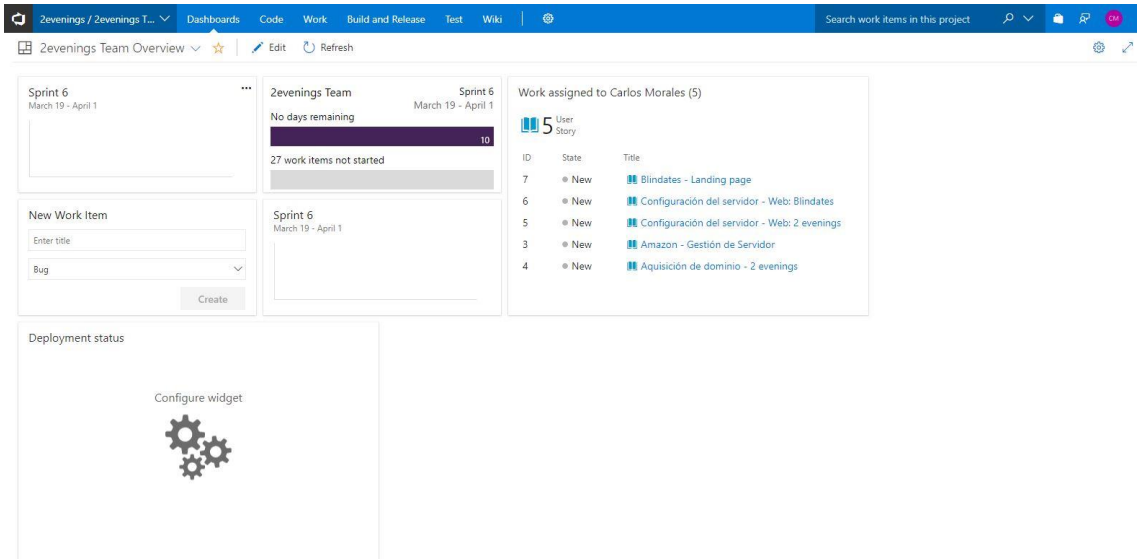

Figura 9. Captura de la herramienta Team Foundation Server.
(Fuente propia)

6. Planificación

Siguiendo la metodología Scrum, realizaremos una planificación basada en *Sprints* de dos semanas, donde incluiremos todos los elementos en lo que trabajaremos. Estos *Sprints* contendrán elementos de desarrollo individuales, que nos permitirán desplegar una nueva versión del proyecto cada final de ciclo que incluirá nuevas funcionalidades y ampliará el alcance del proyecto.

Además, cada uno de los elementos tendrá una estimación de tiempo asociada, calculada anteriormente mediante el análisis previo del trabajo a desarrollar. Este análisis, sumado a la estimación al alza del proyecto en general realizada al comienzo de éste, nos dará una idea muy precisa del tiempo que la tarea nos tomará. A este tiempo se le añadirá la contingencia anteriormente mencionada del 20%.

Al final de cada ciclo de desarrollo, realizamos el cálculo de precisión que hemos tenido a la hora de estimar cada uno de los elementos a desarrollar. Esto nos irá dando un *feedback* progresivo de la contingencia que debemos incluir en cada una de las tareas y modificarla para ser más preciso a la hora de planificar el trabajo a realizar.

Incluiremos en el apartado de Desarrollo la primera iteración de estimaciones del proyecto, donde podremos visualizar en detalle los elementos sobre los que trabajaremos y el tiempo que cada uno nos llevará. Con estos datos podremos construir un diagrama temporal donde veremos cómo evolucionarán las aplicaciones en funcionalidades hasta el día de su compleción.

7. Análisis y especificación

En base a los objetivos propuestos y con el fin de ofrecer un servicio fácilmente accesible, escalable y con características sociales, se ha optado por desarrollar una aplicación multiplataforma que permitirá al usuario acceder desde cualquier sistema móvil. Dicha aplicación seguirá el modelo clásico de cliente-servidor. Además, implementaremos una aplicación web que permitirá la gestión de la información a los restaurantes que formen parte de Blindates. Para el desarrollo de las especificaciones se ha utilizado el sistema IEEE830.

7.1. Características de los usuarios

Tipo de usuario	Usuario Standard
Formación	N/A
Habilidades	Conocimientos informáticos Básicos.
Actividades	Acceso a la generación de las citas y a lo derivado de ello.

Tipo de usuario	Restaurante
Formación	N/A
Habilidades	Conocimientos informáticos Básicos.
Actividades	Acceso a la gestión de su restaurante dentro del sistema.

7.2. Requerimientos del sistema

7.2.1. Requisitos funcionales

7.2.1.1. Blind Dates

Identificador	RF-USR-01
Actor involucrado	Usuario
Nombre	Registro
Descripción	Cualquier usuario podrá registrarse en la aplicación
Requisitos	El usuario proporcionará su número de teléfono como clave fundamental, acompañado de sus datos personales tales como: nombre, apellidos, fecha de nacimiento, lugar de residencia…

Identificador	RF-USR-02
Actor involucrado	Usuario
Nombre	Iniciar sesión
Descripción	Un usuario registrado podrá acceder a la aplicación
Requisitos	El usuario proporcionará sus credenciales para el inicio de sesión: nombre de usuario o número de teléfono, y contraseña

Identificador	RF-USR-03
Actor involucrado	Usuario
Nombre	Cerrar sesión
Descripción	Un usuario registrado podrá cerrar su sesión en la aplicación
Requisitos	Se eliminarán los datos almacenados de la sesión del usuario. El usuario deberá introducir nuevamente sus credenciales para acceder al sistema

Identificador	RF-USR-04
Actor involucrado	Usuario
Nombre	Formulario
Descripción	Un usuario registrado debe completar el formulario de personalidad para poder disfrutar de sus funcionalidades
Requisitos	El usuario debe haberse registrado en la aplicación y haber iniciado la sesión. El formulario se mostrará en pantalla, y tendrá la posibilidad de completarlo en el momento o navegar por la aplicación y completarlo más tarde. En este último caso, el usuario no tendrá acceso a las funciones principales del sistema

Identificador	RF-USR-05
Actor involucrado	Usuario
Nombre	Generar cita
Descripción	Un usuario puede generar citas. Dentro de la cita podrá seleccionar fecha y restaurante dentro de las posibilidades que ofrezca la aplicación en base a la hostelería suscrita y el lugar de la cita
Requisitos	El usuario debe haber completado el formulario de personalidad. El sistema proporcionará los datos de la hostelería registrada en la proximidad del lugar en función de la configuración de proximidad del usuario, dando opción a cambiar de restaurante, de ciudad, día y hora. Una vez se hayan establecido los detalles de ésta la cita será generada

Identificador	RF-USR-06
Actor involucrado	Usuario
Nombre	Aceptar una cita
Descripción	Un usuario que ha sido elegido por compatibilidad para una cita podrá aceptar una cita
Requisitos	El usuario recibirá una notificación de que ha sido seleccionado para una cita, o bien, si la cita ha sido modificada. Se le mostrarán los detalles de la cita. Si el usuario está de acuerdo con el lugar y la fecha señalada, podrá aceptar la cita

Identificador	RF-USR-07
Actor involucrado	Usuario
Nombre	Modificar una cita
Descripción	Un usuario podrá modificar los datos de una cita aún no aceptada
Requisitos	Los detalles de la cita podrán ser modificados y la cita quedará en estado pendiente hasta ser revisada por el otro usuario candidato a la cita. Una vez la cita haya sido aceptada no podrá sufrir modificaciones.

Identificador	RF-USR-08
Actor involucrado	Usuario
Nombre	Listar citas
Descripción	El usuario tendrá acceso a un listado de las citas que ha generado y en las que ha sido elegido
Requisitos	En el listado mostraremos el estado en el que se encuentra la cita con los detalles de la misma. Mostraremos también si se trata de una cita que haya generado el propio usuario o una en la que ha sido seleccionado

Identificador	RF-USR-09
Actor involucrado	Usuario
Nombre	Cancelar cita
Descripción	Un usuario registrado podrá cancelar o denegar una cita
Requisitos	Cuando el usuario recibe la notificación de que ha sido seleccionado para una cita podrá denegar la cita, tanto en el primer momento como en las modificaciones recibidas. El usuario que ha generado la cita también podrá denegar la cita si recibe alguna modificación. El usuario podrá cancelar una cita una vez haya sido creada a través del listado de las citas. Las citas solo se podrán cancelar si no han sido canceladas previamente y no se han realizado todavía

Identificador	RF-USR-10
Actor involucrado	Usuario
Nombre	Valorar una cita
Descripción	Un usuario podrá valorar una cita una vez se haya realizado
Requisitos	Una vez llegados el día y la hora de la cita, se mostrará al usuario un formulario de valoración de la cita. Definiremos una serie de preguntas y una valoración tipo estrella. La información aquí recogida nos dará información sobre el posible mal uso de la aplicación de parte de usuarios, restaurantes. Además recibimos una valoración de compatibilidad de entre los usuarios obteniendo datos sobre nuestro algoritmo de elección

Identificador	RF-USR-11
Actor involucrado	Usuario
Nombre	Listar restaurantes
Descripción	Un usuario podrá listar los restaurantes existentes en la aplicación
Requisitos	El usuario podrá visualizar una lista de restaurantes registrados en el sistema y sus correspondientes fichas sin necesidad de que exista una cita en proceso de generación. El usuario podrá filtrar los resultados por ciudad, nombre, categoría y precio

Identificador	RF-USR-12
Actor involucrado	Usuario
Nombre	Cancelar cuenta
Descripción	Cualquier usuario registrado podrá cancelar su cuenta
Requisitos	Un usuario tendrá acceso a la configuración del sistema para darse de baja. Siguiendo las indicaciones de la nueva legislación en lo referente a la gestión de datos de usuarios de carácter personal, los datos serán completamente eliminados del sistema. Será necesario que el usuario se registre completamente de nuevo en el sistema para poder acceder nuevamente

Identificador	RF-USR-13
Actor involucrado	Usuario
Nombre	Modificar datos de la cuenta
Descripción	Cualquier usuario registrado podrá modificar los datos relativos a la cuenta
Requisitos	El usuario podrá acceder a la zona de configuración, donde encontrará las diferentes pantallas que le permitan modificar sus datos en el sistema: datos personales, datos bancarios, credenciales de acceso…

Identificador	RF-USR-14
Actor involucrado	Usuario
Nombre	Cambiar contraseña
Descripción	Cualquier usuario registrado podrá modificar la contraseña para el acceso de su usuario al sistema
Requisitos	El usuario proporcionará la contraseña actual establecida para su acceso, y la nueva contraseña por la que desea reemplazarla. Será necesaria la confirmación de la nueva contraseña.

Identificador	RF-USR-15
Actor involucrado	Usuario
Nombre	Gestionar usuarios bloqueados
Descripción	Cualquier usuario registrado podrá gestionar los bloqueos hacia posibles candidatos en las citas generadas
Requisitos	El usuario tendrá la opción de bloquear a otros usuarios con el fin de evitar que se generen posibles citas entre ambos. El modo inicial de bloqueo será mediante el número de teléfono. Se estudiará la posibilidad de conectar otras redes sociales con nuestra aplicación y tramitar bloqueos desde allí también

7.2.1.2. Aplicación web de administración

Identificador	RF-RST-01
Actor involucrado	Restaurante
Nombre	Registro
Descripción	Un usuario designado por el restaurante (en adelante, encargado) podrá registrar a éste en la aplicación
Requisitos	El encargado proporcionará los datos legales del restaurante con el fin de poder verificarlo: CIF, domicilio legar, datos personales del autorizado, datos bancarios…

Identificador	RF-RST-02
Actor involucrado	Restaurante
Nombre	Iniciar sesión
Descripción	Un encargado registrado podrá acceder al sistema
Requisitos	El encargado proporcionará los credenciales para el inicio de sesión: CIF de la empresa/nombre de usuario y contraseña

Identificador	RF-RST-03
Actor involucrado	Restaurante
Nombre	Gestionar fotografía
Descripción	Un encargado registrado gestionará la fotografía referente al restaurante
Requisitos	El encargado cargará el archivo de la fotografía en nuestro sistema que será usado en lo referente a los detalles del restaurante para nuestra aplicación

Identificador	RF-RST-04
Actor involucrado	Restaurante
Nombre	Gestionar publicación
Descripción	El encargado será capaz de crear, modificar y eliminar en su tarjeta personal la publicación referida a su oferta y menú.
Requisitos	El encargado tendrá acceso a un modelo de tarjeta común, donde le daremos la posibilidad de incorporar su propio texto descriptivo con una imagen destacada. Además definirá el menú que ofrece para las citas en su local

Identificador	RF-RST-05
Actor involucrado	Restaurante
Nombre	Listar citas
Descripción	Un encargado podrá listar las citas relativas a su restaurante
Requisitos	El encargado visualizará una lista de las citas realizadas y a realizar en su restaurante. Así mismo visualizará su estado. Las citas aparecerán una vez los usuarios hayan aceptado las modificaciones en caso de haberlas

Identificador	RF-RST-07
Actor involucrado	Restaurante
Nombre	Cancelar citas
Descripción	Un encargado podrá cancelar una cita relativa a su restaurante
Requisitos	El encargado tendrá la posibilidad de cancelar alguna de las citas listadas. No se podrán cancelar las citas que han sido canceladas previamente o que ya se han realizado. El restaurante se comprometerá a que esto suceda el menor número de veces posible, aplicando una penalización sobre la suscripción en caso de superar un máximo. Se mostrará a los usuarios pertenecientes a la cita una notificación de modificación, donde el primero en sugerir los nuevos detalles será el que haya generado la cita

Identificador	RF-RST-08
Actor involucrado	Restaurante
Nombre	Modificar disponibilidad
Descripción	Un encargado podrá modificar la disponibilidad del restaurante para celebrar citas
Requisitos	El encargado tendrá acceso a un calendario dónde podrá añadir de forma iterativa, o bien para días puntuales, la disponibilidad del restaurante para celebrar una cita. En caso de que se realice una modificación sobre un día en los que haya una cita programada, y ésta supone la cancelación o modificación de la misma, aplicaríamos la penalización relativa al caso anterior y la notificación a los afectados con sus correspondientes modificaciones sobre los detalles de la cita

Identificador	RF-RST-09
Actor involucrado	Restaurante
Nombre	Modificar datos de la cuenta
Descripción	El encargado podrá modificar los datos del registro
Requisitos	El usuario tendrá acceso a la modificación de los datos introducidos para la suscripción del restaurante

Identificador	RF-RST-10
Actor involucrado	Restaurante
Nombre	Cancelar suscripción
Descripción	El usuario podrá cancelar la suscripción del restaurante
Requisitos	El usuario tendrá acceso a la cancelación de su suscripción para con la aplicación. En caso de que afecte a futuras citas no realizadas, el restaurante deberá abonar la penalización relativa a los casos anteriores y la notificación a los afectados con sus correspondientes modificaciones sobre los detalles de la cita. La cancelación se hará efectiva en la fecha del próximo pago, no haciéndose efectivo el mismo

7.2.2. Requisitos no funcionales

Identificador	RNF-01
Nombre	Diseño de tamaño adaptable
Tipo	Requisito
Fuente del requisito	El diseño de la web se adaptará según la pantalla en que se visualice
Prioridad	Alta/Esencial

Identificador	RNF-02
Nombre	Usabilidad y accesibilidad
Tipo	Restricción
Fuente del requisito	Para aumentar la usabilidad de la aplicación se utilizarán colores con alto contraste
Prioridad	Baja/Opcional

Identificador	RNF-03
Nombre	Navegador Web
Tipo	Requisito
Fuente del requisito	La aplicación se podrá visualizar en cualquiera de los principales navegadores del mercado
Prioridad	Alta/Esencial

Identificador	RNF-04
Nombre	Control de versiones
Tipo	Restricción
Fuente del requisito	Se necesita de un sistema de control de versiones para trabajar sobre las diferentes iteraciones de nuestro desarrollo
Prioridad	Alta/Esencial

Identificador	RNF-05
Nombre	Escalabilidad
Fuente	El sistema debe soportar grandes picos de consultas simultáneas a la base de datos. Para ello descargaremos responsabilidad sobre los dispositivos del usuario, buscando el balance entre el rendimiento del servidor y la experiencia de usuario
Prioridad	Alta/Esencial

Identificador	RNF-06
Nombre	Seguridad sobre el acceso la información
Fuente	Se desarrollará un sistema de servicios y accesos que no permitan a ningún agente externo no identificado a realizar consultas ni modificaciones sobre la información que se encuentra en nuestro sistema.
Prioridad	Alta/Esencial

8. Diseño

BlinDates se compone de tres elementos principales: aplicación móvil, página de administración (*back office*) y API, servicios que nos proporcionarán la información necesaria para mostrar en las aplicaciones. Diseñaremos entonces un completo sistema que nos permita gestionar todas las características de cada uno de los elementos.

8.1. Diseño de la arquitectura de servidores

Este será el comienzo de nuestro proyecto. Aunque el elemento principal de nuestro sistema sea la aplicación móvil, será necesaria la configuración de un servidor que almacenará los diversos elementos de los que la aplicación necesita.

El servidor alojará en primer lugar nuestra página web (*landing page*). Para ello será necesario la adquisición del dominio y la configuración del servidor DNS para que resuelva la dirección IP del dominio dentro de nuestro sistema. En nuestro caso, el dominio adquirido será www.2evenings.es que pertenecerá a la "empresa" desarrolladora de nuestro proyecto. Sobre este dominio, configuraremos un subdominio que alojará nuestra aplicación BlinDates: blindates.2evenings.es. La página web que alojaremos en ese dominio, además de funcionar como *landing page*, será la puerta de acceso a nuestro sistema de *back office* o portal de gestión, dónde los restaurantes podrán gestionar la información de su local que aparece en nuestro sistema.

Para ambos elementos (aplicación móvil y página web) se hace necesario un sistema de servicios que nos permita recoger la información almacenada en la base de datos. Aunque entraremos en más detalle sobre este tema más adelante, por el momento identificamos la necesidad de la creación de un nuevo subdominio, sobre el que realizaremos las consultas, tanto desde la aplicación móvil como desde nuestra plataforma de gestión: blindates.api.2evenings.es.

Por último, aunque será analizado en detalle en el momento de la implementación del servidor, será imprescindible en la elección y el estudio de las posibilidades tener en cuenta la escalabilidad del proyecto [], pudiendo aumentar los recursos del servidor cuando sea necesario. BlinDates es un proyecto con un alto porcentaje de público objetivo.

Por tanto, la elección de la plataforma para la implementación de la arquitectura de servidores debe proporcionarnos las herramientas necesarias para que al comienzo de nuestro proyecto podamos integrar un sistema de rendimiento medio y coste reducido, pero que conforme a la evolución del proyecto y a la evolución en el número de suscripciones y de usuario activos, seamos capaces de cambiar dichas configuraciones con el fin de adaptarnos a las nuevas necesidades sin que esto afecte al producto.

Partiendo entonces de todo lo mencionado anteriormente, son pocas las posibilidades de elección sobre la plataforma de servidor a contratar. Aunque bien es cierto que son muchas las plataformas existentes (1&1MiWeb [16], Hostalia [17] , Hostinger [18]…), prácticamente todas

ellas carecen de acceso vía consola o conexión remota al servidor, lo cual se hace prácticamente necesario para trabajar sobre las diferentes configuraciones a realizar a nivel de servidor web, permitiéndonos la creación y gestión de los diferentes dominios ya mencionados, y la inclusión de paquetes y librerías para las aplicaciones web y API que hospedará.

Amazon Sever [19] será la elección que realizaremos para la contratación de nuestro servidor. Esta plataforma nos permite, con un coste ínfimo mensual basado en el tráfico generado sobre la instancia, configurar los diferentes elementos que son necesarios para la completa arquitectura.

Además de la creación de diferentes servidores (en nuestro caso solo será uno), nos proporciona herramientas tales como Route 53 [20] dándonos completo acceso al servidor DNS donde podremos configurar los diferentes dominios y subdominios necesarios, creación de IP elástica que nos proporciona una IP pública fija sin coste alguno, o Simple Email Service [21], que nos otorga un sistema de *mailing* configurable en nuestro sistema.

Haciendo referencia a la escalabilidad mencionada anteriormente, Amazon nos brinda la posibilidad de cambiar el tipo de instancia una vez creada. Esto quiere decir que podremos otorgar diferentes valores de espacio, memoria RAM o velocidad de procesador a nuestra instancia cuando el sistema lo requiera sin que esto afecte a nuestro producto ni a su funcionamiento.

Por último, a la hora de la creación de la instancia del servidor, son varias las posibilidades de elección sobre el sistema operativo. Basándonos de nuevo en los criterios antes mencionados, será Linux el sistema que escojamos para nuestra instancia. Aunque Amazon nos ofrece la posibilidad de crear servidores Windows, las licencias de software encarecen la creación y el uso de esta, y, dado que no hacemos uso de ninguna librería o *framework* nativo de Windows, esto nos abre las puertas a un SO libre de pago por licencias y abierto a la configuración que necesitamos.

8.2. Diseño de experiencia de usuario: casos de uso

Para especificar la comunicación y el comportamiento del sistema, se necesita saber cómo va a ser su interacción con los usuarios y/u otros sistemas. Los casos de uso nos muestran de forma gráfica la relación que mantienen los actores con otros actores y con el sistema.

8.2.1. Aplicación de usuario

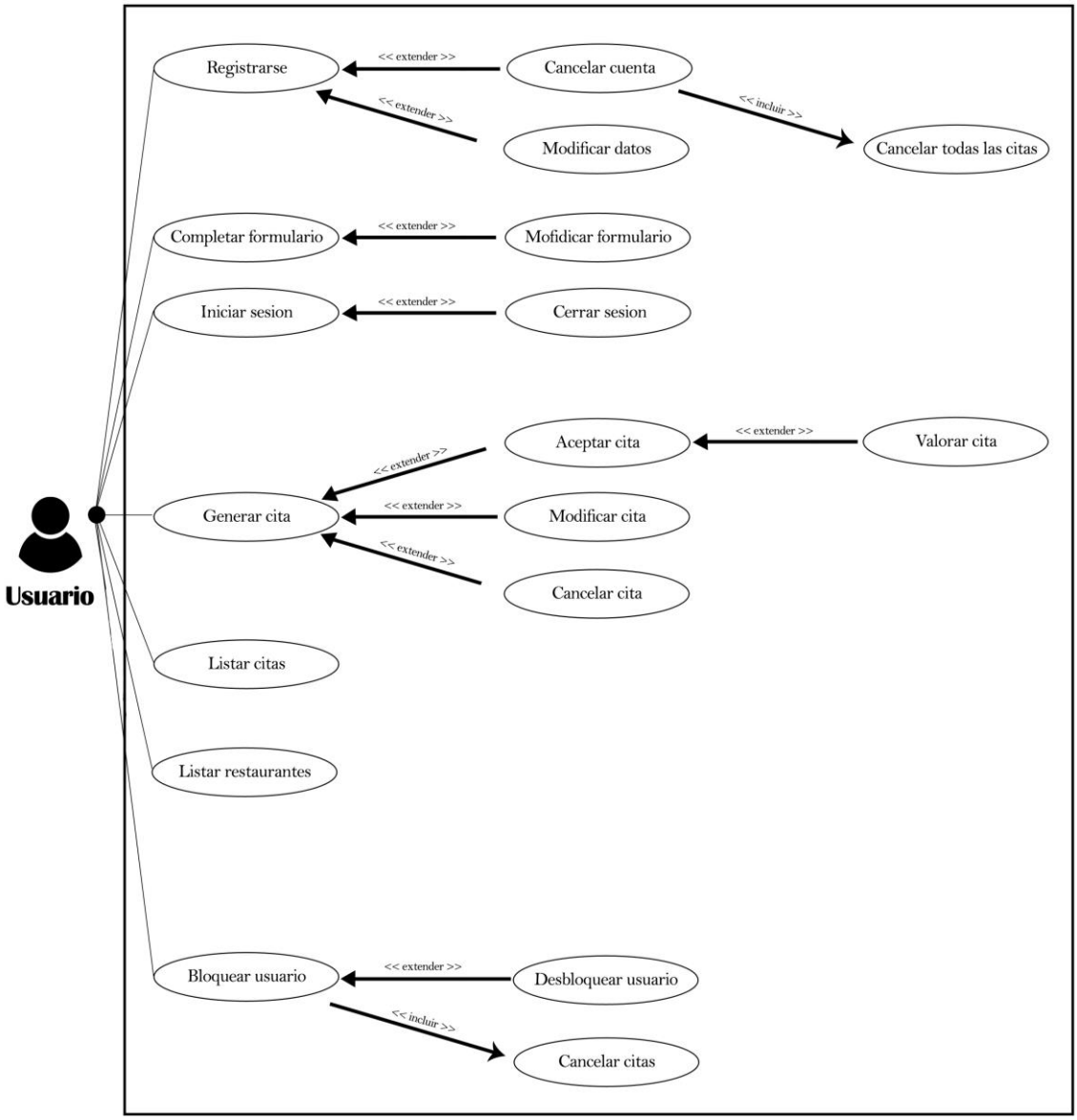

Figura 10. Casos de uso: aplicación móvil
(Fuente propia)

- [RF-USR-01] **Registro**: Al acceder a la aplicación el usuario podrá registrarse en el sistema a través de un formulario donde se recogerán sus datos personales para identificarlo.

- [RF-USR-02] **Iniciar sesión**: Al acceder a la aplicación el usuario podrá introducir sus credenciales de registro para identificarse en el sistema.

- [RF-USR-04] **Completar formulario**: Al completar el registro el usuario accederá al formulario de personalidad sobre el que basamos la búsqueda de compatibilidad a la hora de generar citas. El usuario podrá completarlo en el momento o continuar navegando por la aplicación y completarlo más adelante. La cuenta no estará activa hasta que el formulario no haya sido completado.

- [RF-USR-13] **Modificar datos**: Una vez identificado en el sistema, el usuario podrá modificar sus datos personales. Está opción estará recogida en el panel de configuración de la herramienta, que será accesible a través de uno de los botones de navegación.

- [RF-USR-04] **Modificar formulario**: Una vez identificado en el sistema, el usuario podrá modificar sus respuestas sobre el formulario de personalidad. Está opción estará recogida en el panel de configuración de la herramienta, que será accesible a través de uno de los botones de navegación.

- [RF-USR-03] **Cerrar sesión**: Una vez identificado en el sistema, el usuario podrá desacreditarse eliminando su sesión en éste. Está opción estará recogida en el panel de configuración de la herramienta, que será accesible a través de uno de los botones de navegación.

- [RF-USR-12] **Cancelar cuenta**: Una vez identificado en el sistema, el usuario podrá cancelar su cuenta con el fin de interrumpir la interacción del usuario con el sistema. Sus datos no serán utilizados de aquí en adelante. Se notificará al usuario que, si así lo desea, sus datos serán eliminados del sistema siguiendo la normativa vigente que establece el GDPR. Si esto ocurre, será necesario un registro completo en el sistema en caso de querer acceder nuevamente.

- [RF-USR-05] **Generar una cita**: Una vez identificado en el sistema y habiendo completado el formulario de personalidad, el usuario podrá generar una cita a ciegas. Está opción se recogerá en la pantalla principal del sistema a través de un botón. El sistema pedirá los detalles de la cita (fecha y lugar) al usuario, y una vez éstos hayan sido aceptados, el sistema procederá a encontrar al candidato por medio de un logaritmo de compatibilidad.

- [RF-USR-06] **Aceptar una cita**: Habiendo completado el formulario de personalidad, el usuario podrá ser seleccionado como candidato para una cita generada. El usuario recibirá una notificación sobre la nueva cita añadida, y dentro del sistema podrá ver los detalles de ésta y aceptarla. La cita será añadida a su listado de próximas citas.

- [RF-USR-07] **Modificar una cita**: Si los detalles de la cita no son los deseados, el usuario tendrá la opción de modificarlos a su gusto y reenviar de nuevo los detalles al otro candidato. Éste a su vez podrá aceptar la cita o modificarla de nuevo.

- [RF-USR-09] **Cancelar una cita**: El usuario podrá cancelar la cita, siendo ésta eliminada para los dos candidatos participantes y notificando al segundo usuario de que la cita ha sido cancelada.

- [RF-USR-10] **Valorar una cita**: Habiendo tenido lugar una cita generada en el sistema, el sistema mostrará al usuario un formulario de valoración donde se recogerán datos sobre la cita: si se ha producido correctamente, la valoración sobre la compatibilidad con el otro usuario y la valoración del restaurante donde la cita ha tenido lugar.

- [RF-USR-08] **Listar citas**: El usuario podrá acceder a través de una de las opciones de navegación al listado de citas asociadas a él, ya sea generadas por él mismo o citas donde ha sido elegido por compatibilidad.

- [RF-USR-11] **Listar restaurantes**: El usuario podrá acceder a través de una de las opciones de navegación al listado de restaurantes disponibles en el sistema. Este listado podrá ser filtrado por diversos campos de búsqueda: nombre, distancia, localidad…

- [RF-USR-15] **Bloquear usuario**: El usuario podrá bloquear otros usuarios del sistema con el fin de eliminar la posibilidad de, al generar una cita, el sistema elija a este segundo usuario como usuario de máxima compatibilidad, y así evitar tener una cita a ciegas con dicha persona. El bloqueo se realizará en primera instancia a través del número de teléfono, siendo clave para el registro de los usuarios. Si el usuario bloqueado había sido previamente elegido para alguna de las citas generadas en futuras fechas, estas citas serán canceladas de forma automática y se notificará a ambos usuarios.

- [RF-USR-15] **Desbloquear usuario**: El usuario podrá desbloquear a los usuarios que previamente hayan sido bloqueados.

8.2.2. Aplicación de restaurante

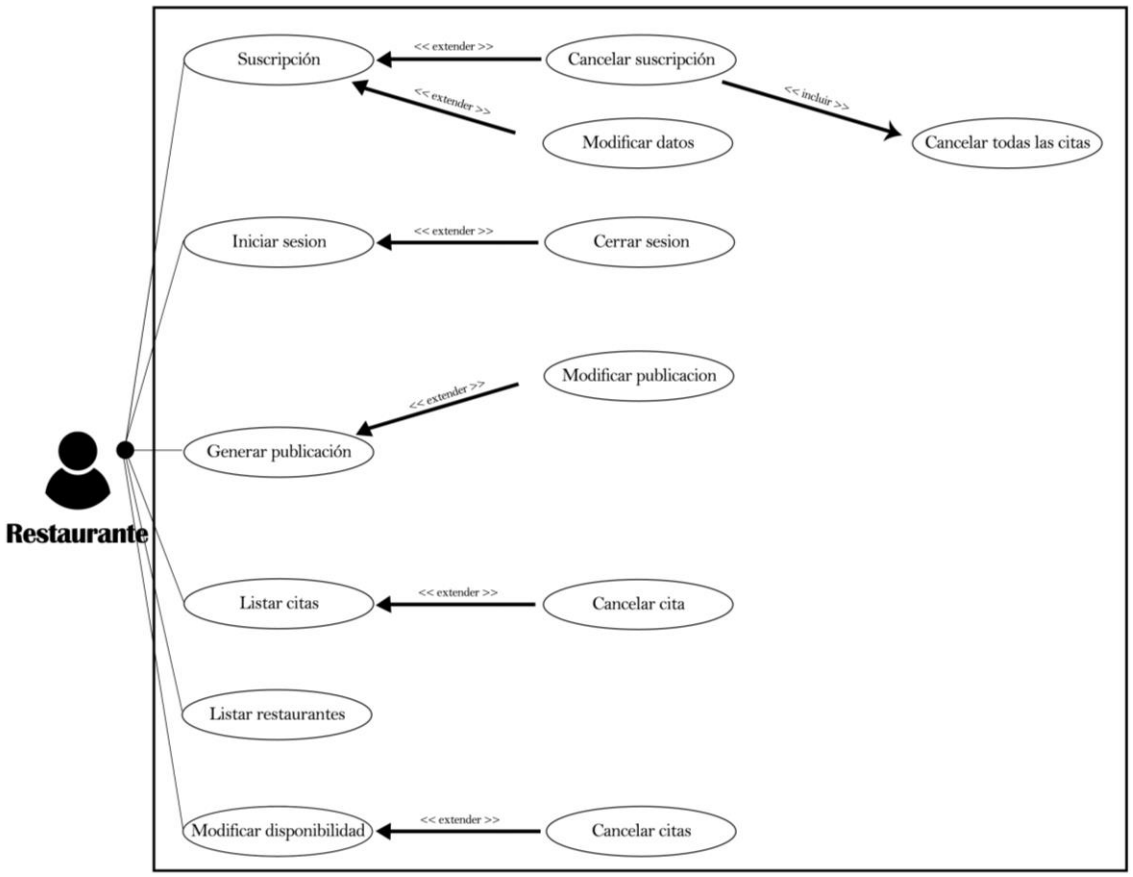

Figura 11. Casos de uso: back office
(Fuente propia)

- [RF-RST-01] **Registro**: Al acceder a la aplicación el encargado podrá registrar en el sistema a un restaurante a través de un formulario donde se recogerán los datos del restaurante para identificarlo.

- [RF-RST-02] **Iniciar sesión**: Al acceder a la aplicación el encargado podrá introducir sus credenciales de registro para identificarse en el sistema.

- [RF-RST-09] **Modificar datos**: El encargado podrá modificar los datos relativos al registro del restaurante.

- [RF-RST-10] **Cancelar suscripción**: El encargado podrá cancelar su cuenta con el fin de interrumpir la interacción del restaurante con el sistema. Sus datos no serán utilizados de aquí en adelante. La baja se hará efectiva en el momento inmediato de la solicitud, y se cancelarán de forma automática todas las citas existentes en el sistema.

- [RF-RST-04] **Generar publicación**: Una vez identificado en el sistema, el encargado podrá generar la tarjeta descriptiva sobre su restaurante con el fin de aparecer en las opciones de lugar dentro de las citas generadas. Esta ficha recogerá una foto, una pequeña descripción y las diferentes posibilidades dentro del menú. Todos los menús de nuestro sistema tendrán un precio fijado de 15€.

- [RF-RST-04] **Modificar publicación**: Habiendo generado la publicación de su ficha previamente, el encargado podrá modificar cualquiera de los datos que ésta recoge y guardar las modificaciones.

- [RF-RST-05] **Listar citas**: El encargado podrá acceder al listado de citas generadas que han elegido su restaurante como lugar donde hacer efectiva la cita. En este listado se mostrarán los detalles de las citas, y, para las que todavía no hayan tenido lugar, la opción de cancelación de éstas.

- [RF-RST-08] **Modificar disponibilidad**: La disponibilidad publicada podrá ser modificada. Si en la modificación sufrida, alguna de las citas ya generadas se ve afectada, será cancelada de forma automática y los usuarios serán notificados.

8.3. Diseño de la persistencia

El almacenamiento de datos es sin duda alguna el aspecto más complejo que abordamos en nuestro sistema. La interconexión que componen nuestras aplicaciones requiere que la base de datos sea un sistema robusto y que los elementos principales queden bien definidos desde el comienzo. Un mal análisis del diseño podría llevarnos a tener que rehacer grandes partes de nuestro sistema por no haber tenido en cuenta algún detalle.

Por tanto, seguiremos todos y cada uno de los casos de uso que hemos identificado anteriormente para cada requisito funcional, sirviéndonos de base para construir nuestro modelo de base de datos. Integraremos en un único esquema todas las entidades relacionadas para los dos tipos de usuarios: restaurantes y candidatos a citas.

El siguiente diagrama nos muestra todas las entidades que hemos identificado como necesarias tras el análisis, con sus correspondientes relaciones entre las entidades. Para todas las relaciones que hallemos, configuraremos las claves ajenas necesarias para que nuestra base de datos identifique inmediatamente cuando una inserción, modificación o eliminación sea errónea.

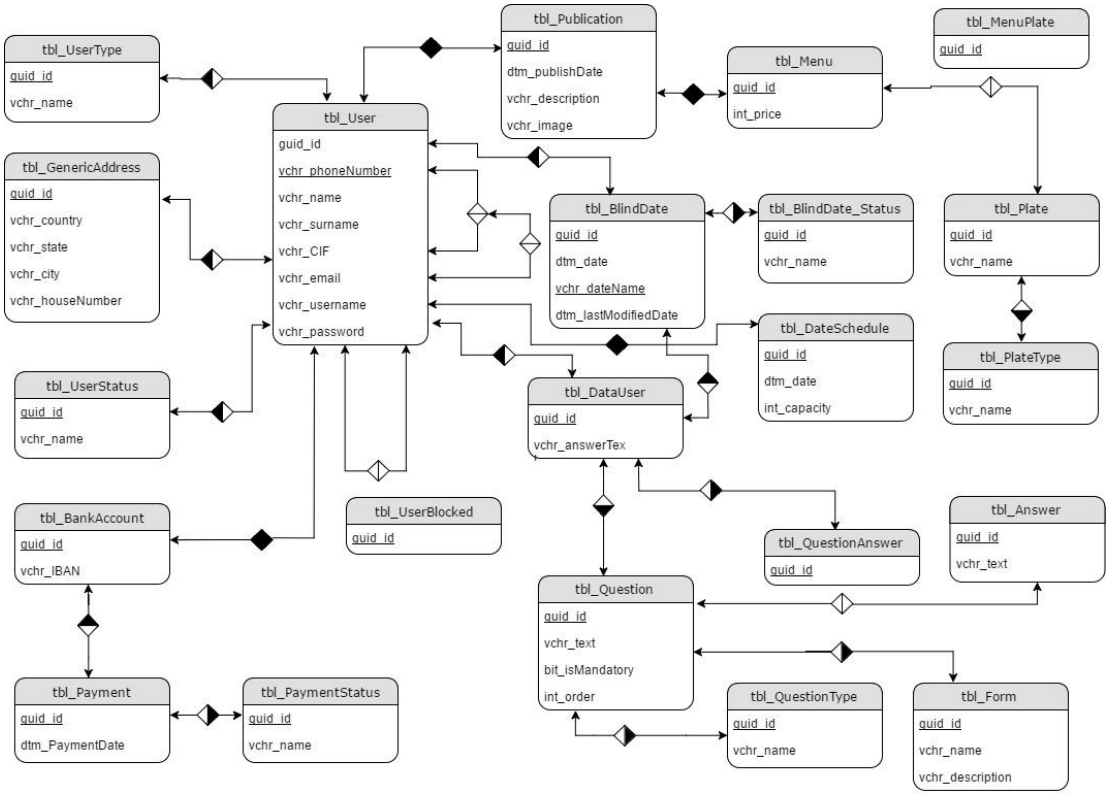

Figura 12. Esquema de la base de datos de BlinDates
(Fuente propia)

Como podemos observar a partir del esquema, la relación entre nuestras entidades jugará un papel importante en la implementación de nuestra persistencia de datos. Esto nos lleva a descartar tecnologías como MongoDB [22], orientado a la información masiva y que suprime la entidad-relación.

8.3.1. Mongo DB

Es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta.

- NoSQL.
- Guarda estructuras de datos en documentos tipo JSON.
- De código abierto.
- Soporta la búsqueda por campos, consultas de rangos y expresiones regulares.
- Tiene un concepto de índices parecido al de los modelos relacionales.
- Favorece la utilización de Javascript del lado del servidor.

Pero dada la funcionalidad de nuestro sistema, se hará necesario el uso de una base de datos tradicional o relacional, donde los datos definidos mantienen un patrón común, dividido en entidades (tablas), y donde crearemos las relaciones entre las diferentes entidades que existen en nuestro sistema.

Es por ello que nos decantaremos por un modelo clásico SQL para el almacenamiento de nuestros datos. Bajo este modelo, son varias las herramientas extendidas en el mundo profesional para su implementación.

8.3.2. Oracle

Se considera a Oracle [23] como uno de los sistemas de bases de datos más completos, destacando: soporte de transacciones, estabilidad, escalabilidad, y soporte multiplataforma. Oracle es la primera compañía de software que desarrolla e implementa software para empresas cien por cien activo por Internet a través de toda su línea de productos: base de datos, aplicaciones comerciales y herramientas de desarrollo de aplicaciones y soporte de decisiones.

- De tipo objeto-relacional.
- Uno de los más potentes y completos.
- Soporta el manejo de grandes volúmenes de datos.
- Estable y escalable.

Sin embargo este producto tiene un precio elevado en el mercado y un alto coste de mantenimiento. Aunque las características de este sistema lo harían idóneo para el desarrollo de nuestra plataforma, la necesidad de un personal cualificado para su manejo y la no existencia de presupuesto destinado al desarrollo lo convierte en inviable (listado de precios de licencia de uso de Oracle [24]).

8.3.3. SQL Server

SQL Server [25] es un sistema de manejo de bases de datos del modelo relacional, desarrollado por la empresa Microsoft.

- Soporte de transacciones.
- Soporta procedimientos almacenados.
- Incluye también un entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información.
- Además permite administrar información de otros servidores de datos.
- Es un sistema escalable, estable y seguro.

En esta ocasión encontramos un problema de compatibilidad para la elección de este sistema. SQL Server actualmente se encuentra disponible únicamente para sistemas operativos de Microsoft, ya que son los desarrolladores del software. Así pues, aunque sea la elección a tomar una vez el proyecto evolucione y comience a crecer, la implementación de un segundo servidor dedicado exclusivamente al almacenamiento de datos no entra de los planes del proyecto piloto.

8.3.4. MySQL

MySQL [26] es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como la base datos *Open Source* más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

- Relacional, multi-hilo y multiusuario.
- Potente
- Simple
- Fácil aprendizaje.
- Buena integración con otros lenguajes.
- De código abierto.

Aunque por detrás de las herramientas anteriormente mencionadas en cuanto a potencia y posibilidades, MySQL nos brinda la posibilidad de gestionar la información en un entorno de desarrollo web multiplataforma de forma gratuita. No necesita de grandes requerimientos de sistema, y, gracias a una configuración sencilla y una buena compatibilidad con lenguajes y *frameworks*, la hacen idónea para nuestro proyecto.

8.3.5. Logs

Dentro de este apartado no podemos dejar de lado los archivos de registro que serán almacenados en nuestro servidor. En nuestro caso, generaremos para cada uno de los entornos web implementados en nuestro servidor (*back office y API*) dos archivos:

- ***access.*.log***: este archivo creará un registro de cada llamada que se ha hecho a cualquiera de los archivos que compone la web. En el registraremos el dispositivo que realiza la llamada (navegador), y la IP desde la cual está accediendo a nuestro sistema. Esto nos permitirá poder identificar patrones de uso, nos otorga datos para un estudio de mercado, y nos permite identificar algún posible error en primera instancia.
- ***.log***: este archivo nos informará únicamente de los errores sucedidos en nuestro sistema. Desde los errores que provengan del propio servidor por problemas como espacio o velocidad de procesamiento, hasta el manejo de errores más específico que se pueda referenciar en los diferentes servicios web.

Estos archivos se almacenarán en nuestro servidor, y serán generados nuevos archivos diariamente, incluyendo la fecha en formato YYYYMMDD en el nombre de los archivos, lo que nos permitirá identificar más rápidamente y tener un mayor control sobre errores aislados.

8.3.6. Auditoría

Por último, dentro del sistema de datos, es necesario tratar la auditoría de estos. Si bien es cierto que es un tema sensible con la última legislación en rigor desde el pasado 25 de mayo, cumpliremos a rajatabla la normativa y los principios básicos de esta.

La auditoría de los datos no será un requisito para el prototipo piloto a desarrollar. En esta primera fase trabajaremos únicamente con la base de datos anteriormente definida. Sin embargo, al igual que con la evolución y crecimiento del proyecto será necesaria la implementación de un servidor exclusivo para el manejo de datos, con la puesta en marcha de BlinDates será necesaria la creación de una segunda base de datos, ampliación de la ya existente, donde llevaremos un registro de todas las acciones y los cambios que sufran las entidades en nuestro sistema a lo largo de su uso.

Con esta base de datos como herramienta, su análisis nos podrá proporcionar información del uso que realizan de nuestra aplicación los usuarios, con finalidades como la corrección de posibles errores identificados o la penalización de aquellos usuarios que hagan un mal uso de la aplicación.

8.4. Diseño de la arquitectura conceptual

El diseño de nuestra arquitectura, como anteriormente hemos mencionado, se compondrá de tres elementos principales, añadiendo ahora un cuarto elemento: la persistencia de datos. Así, nuestra API será la encargada de atacar la base de datos, recibiendo las peticiones realizadas bien por la página de administración, bien por la aplicación móvil.

Cabe mencionar que la API contendrá un módulo de autenticación, el cual será participe en cada una de las llamadas con el único fin de asegurar al sistema que la petición se realiza de manera segura y controlada, evitando así posibles escapes de datos o ataques a nuestro sistema.

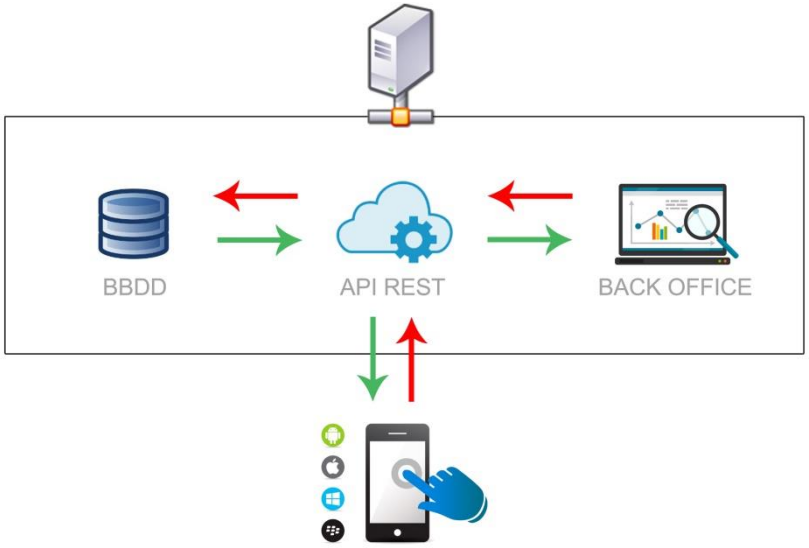

Figura 13. Diseño arquitectura BlinDates
(Fuente propia)

8.5. Diseño de la API Rest

Como podemos observar a partir del esquema de la arquitectura, nuestra API jugará un papel fundamental en el proyecto completo. Será el puente de conexión hacia la base de datos para ambas aplicaciones. Por eso, debemos realizar una buena definición del sistema y un diseño que nos permita que no se crucen los diferentes servicios, pero sean reusables en caso de ser necesario. Es decir, realizaremos un diseño de la API que aislará los servicios de una aplicación sobre la otra, pero incluiremos un puente de conexión entre los servicios para la reutilización de alguna de las funcionalidades si se diese el caso.

El porqué de la implementación de la API en modelo REST, dejando a un lado el que sea una tendencia actual y muy extendida, viene explicado por el hecho de que, al realizarse un servicio REST entre dos proyectos cualesquiera, el servicio pierde todos sus datos. Es decir, al enviar datos a través de una petición al servicio, no recuerda estos datos en la siguiente petición. Al implementar un modelo REST, no será necesario mantener ningún estado de información dentro de nuestra API, lo que supondrá una mejora del rendimiento de los servicios web y simplificará el diseño e implementación de los componentes del servidor, ya que elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

Método	Función	Descripción
GET	http://blindates.api.2evenings.es/user/login	Comprueba los datos de acceso de usuario y devuelve el usuario los credenciales son correctos.
POST	http://blindates.api.2evenings.es/user/register	Añade un nuevo usuario con los datos introducidos.
GET	http://blindates.api.2evenings.es/users	Devuelve todos los usuarios del sistema. Puede recibir un parámetro de tipo de usuario.
GET	http://blindates.api.2evenings.es/user/:id/status	Devuelve los estados disponibles en el sistema para los usuarios. Si recibe un usuario por parámetro devolverá el estado de éste.
GET	http://blindates.api.2evenings.es/user/:id/type	Devuelve los tipos de usuario disponibles en el sistema. Si recibe un usuario por parámetro devolverá el tipo de éste.
PUT	http://blindates.api.2evenings.es/user	Actualiza el usuario recibido por parámetro.
GET	http://blindates.api.2evenings.es/user/:id/blockers	Devuelve todos los usuarios bloqueados en el sistema. Si recibe un usuario por parámetro devolverá los bloqueados relacionados con éste.
POST	http://blindates.api.2evenings.es/user/:id/blockers	Añade un nuevo usuario bloqueado para el usuario recibido por parámetro.
DELETE	http://blindates.api.2evenings.es/user/:id/blockers/:blocker	Elimina el usuario bloqueado recibido por parámetro.
GET	http://blindates.api.2evenings.es/user/:address	Devuelve la dirección en base a los datos introducidos.
POST	http://blindates.api.2evenings.es/user/:address	Añade una nueva dirección con los datos introducidos.
GET	http://blindates.api.2evenings.es/user/:id/description	Devuelve la descripción para un usuario recibido por parámetro.
POST	http://blindates.api.2evenings.es/user/:id/description	Añade la descripción para un usuario recibido por parámetro.
GET	http://blindates.api.2evenings.es/restaurant/:type	Devuelve el listado de los restaurantes registrados en nuestro sistema. Filtrará por tipo de restaurante si lo recibe por parámetro.

GET	http://blindates.api.2evenings.es/restaurant/types	Devuelve los tipos de cocina predefinidos para los restaurantes en el sistema
GET	http://blindates.api.2evenings.es/restaurant/:id/	Devuelve los detalles del restaurante recibido por parámetro.
GET	http://blindates.api.2evenings.es/restaurant/:id/menu	Devuelve los menús disponibles en el sistema. Si recibe un usuario por parámetro devolverá el menú relacionado con éste.
POST	http://blindates.api.2evenings.es/restaurant/:id/menu	Añade un nuevo menú para el usuario recibido por parámetro.
GET	http://blindates.api.2evenings.es/restaurant/:id/aviability	Devuelve las reservas disponibles para un restaurante recibido por parámetro.
POST	http://blindates.api.2evenings.es/restaurant/:id/aviability	Añade una nueva reserva disponible para un usuario recibido por parámetro.
DELETE	http://blindates.api.2evenings.es/restaurant/:id/aviability/:aviability	Elimina una reserva disponible para un usuario recibido por parámetro.
GET	http://blindates.api.2evenings.es/plate/types	Devuelve los tipos de plato disponibles en el sistema.
GET	http://blindates.api.2evenings.es/plate/:type/	Devuelve los platos disponibles en el sistema. Si recibe un tipo por parámetro devolverá los platos relacionados con éste.
POST	http://blindates.api.2evenings.es/plate/	Añade un nuevo plato con los datos recibidos por parámetro.
GET	http://blindates.api.2evenings.es/form/:form/answers/:answer	Devuelve todas las respuestas disponibles en el sistema. Si recibe una pregunta por parámetro devolverá las preguntas relacionadas con éste.
GET	http://blindates.api.2evenings.es/form/:form/:user	Devuelve los datos ya almacenados de las respuestas de usuario para el formulario especificado.
POST	http://blindates.api.2evenings.es/form/:user	Añade los datos relacionados con una respuesta y un usuario.
GET	http://blindates.api.2evenings.es/blindate/:user/match	Devuelve el usuario con mayor rango de compatibilidad a través de un algoritmo sobre las respuestas del formulario de personalidad.
POST	http://blindates.api.2evenings.es/blindate/	Añade la cita a ciegas con los datos introducidos.
GET	http://blindates.api.2evenings.es/blindate/:user	Devuelve las citas a ciegas para el usuario recibido por parámetro.
GET	http://blindates.api.2evenings.es/blindate/:blindate	Devuelve los detalles de la cita recibida por parámetro.
GET	http://blindates.api.2evenings.es/blindate/:restaurant/:date/:status	Devuelve las citas a ciegas registradas en un restaurante. Puede recibir parámetro de fecha y de estado para filtrar el resultado.

Tabla 3. Especificación de la API Rest
(Fuente propia)

8.6. Diseño de la arquitectura Back-end

Habiendo definido ya las necesidades y los diferentes servicios que nuestra API implementará, es el momento de analizar las diferentes tecnologías que nos permitan llevarlo a cabo. La arquitectura de una aplicación basada en API REST es indiferente del lenguaje, y, por tanto, cualquier alternativa es suficiente para acometer una tarea de hacer un *back-end* que permita su reutilización para diferentes plataformas. Así mismo, desde plataformas web o móvil es también indiferente en qué lenguaje esté construido el API, por lo que al final nuestra elección tiene bastante flexibilidad.

8.6.1. NodeJS

El desarrollo con NodeJS [27] de un API REST es especialmente indicado, dado que nos permite construir el *back-end* con el mismo lenguaje con el que se construye el *front-end*. Además, dadas las características de esta plataforma, también se hace muy adecuada para el desarrollo REST ya que Node puede atender a más usuarios de manera concurrente. En NodeJS hay frameworks potentes para el desarrollo de APIs y aplicaciones en general, como Express o algunos enfocados al desarrollo de APIs de manera más particular como SailsJS.

Aunque JavaScript tradicionalmente ha sido relegado a realizar tareas menores en el navegador, es actualmente un lenguaje de programación totalmente capaz como cualquier otro lenguaje tradicional. Además, JavaScript tiene la ventaja de poseer un excelente modelo de eventos, ideal para la programación asíncrona.

JavaScript es a día de hoy un lenguaje omnipresente, conocido por millones de desarrolladores. Esto reduce la curva de aprendizaje de NodeJS, ya que la mayoría de los desarrolladores no tendrán que aprender un nuevo lenguaje para empezar a construir aplicaciones.

El principal motivo por el que descartaremos esta tecnología de nuestra elección es que está directamente relacionada con tecnologías de base de datos orientadas a la información masiva, la cual hemos descartado previamente debido a la necesidad de implementar un modelo tradicional de entidad-relación.

8.6.2. .NET

En esta tecnología también existen diversas alternativas sencillas y rápidas para implementar un API. El propio Microsoft nos ofrece ASP.NET Web API, un complemento esencial para construir servicios HTTP especialmente pensados para REST.

Encontramos el primer impedimento al comienzo de nuestro análisis sobre esta tecnología, ya que está desarrollada por la empresa Microsoft, y la implementación de un servidor basado en Linux nos impedirá que nuestro servicio funcione en la máquina. Aunque .NET ha desarrollado una variante llamada .NET Core que nos proporciona varias funcionalidades de la tecnología soportadas en cualquier sistema operativo, el desarrollo de API REST sobre esta nueva tecnología no se hace tan recomendable debido a los posibles problemas de compatibilidades que podamos hallar durante su ejecución.

8.6.3. Ruby on Rails

Ruby on Rails [28] es un armazón para construir aplicaciones web que acceden a bases de datos. Un conjunto de librerías, automatismos y convenciones destinados a resolver los problemas más comunes a la hora de desarrollar una aplicación web, para que el programador pueda concentrarse en los aspectos únicos y diferenciales de su proyecto en lugar de los problemas recurrentes.

Ruby on Rails es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, siguiendo el paradigma del patrón Modelo Vista Controlador (MVC). Trata de combinar la simplicidad con la posibilidad de desarrollar aplicaciones del mundo real escribiendo menos código que con otros *frameworks* y con un mínimo de configuración. El lenguaje de programación Ruby permite la meta programación, de la cual Rails hace uso, lo que resulta en una sintaxis que muchos de sus usuarios encuentran muy legible.

8.6.4. Python

Python [29] es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones de escritorio a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.

Contiene una gran cantidad de librerías, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.

8.6.5. PHP

PHP [30] es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página web resultante. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

Es un lenguaje web y cuenta con un gran número de *frameworks* donde elegir, pero de entre todos nos quedamos con Symfony [31], un framework que se adapta perfectamente a las dimensiones de nuestro proyecto, que nos permite construir rápidamente APIs que se comuniquen con nuestra base de datos y sirvan los datos en formato JSON. Es muy ligero, está bien documentado y, sin duda, una opción muy recomendable cuando el peso de nuestra aplicación recae del lado del cliente.

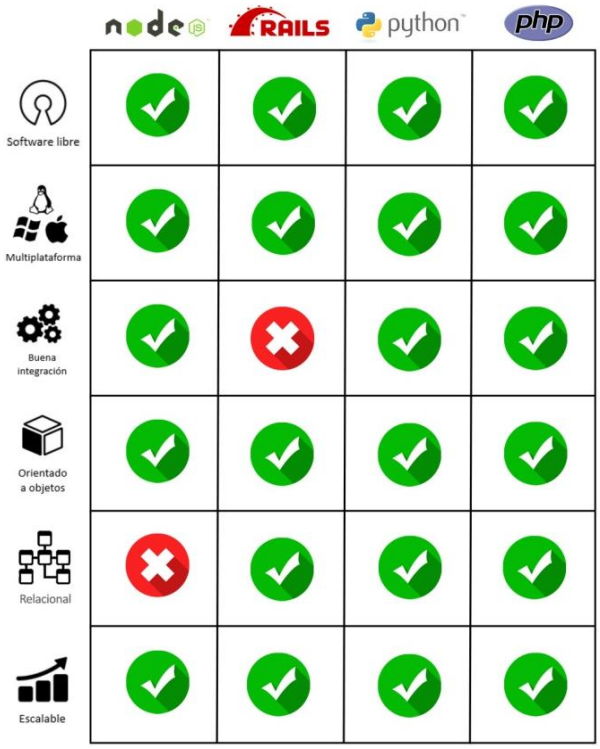

Figura 14. Comparativa de tecnologías back-end
(Fuente propia)

Después de haber realizado el análisis de las diferentes tecnologías, habiéndolas enfrentado en una comparativa, son pocas las diferencias tangibles entre todas nuestras opciones. Para la elección de la tecnología back-end nos decantaremos entonces por una tecnología con la que nos sintamos cómodos trabajando, y una experiencia previa puede ayudarnos a comprender más a fondo la tecnología y lograr un desarrollo más robusto. Así, PHP se convierte en el candidato ideal.

Como hemos mencionado anteriormente, construiremos la API REST basando el desarrollo en uno de los *frameworks* que PHP nos ofrece: Symfony.

8.7. Diseño de la arquitectura Front-end

En diseño de software, el *front-end* es la parte del software que interactúa con el o los usuarios. La separación del sistema en *front-end* y *back-end* es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas. La idea general es que el *front-end* sea el responsable de recolectar los datos de entrada del usuario, que pueden ser de muchas y variadas formas, y los transforma ajustándolos a las especificaciones que demanda el *back-end* para poder procesarlos, devolviendo generalmente una respuesta que el *front-end* recibe y expone al usuario de una forma entendible para este.

En nuestro proyecto diferenciaremos dos bloques en cuanto al *front-end* se refiere. Por un lado hablaremos sobre la arquitectura necesaria para el desarrollo de nuestro *back office*, y en un segundo nivel definiremos las tecnologías con las que desarrollaremos nuestra aplicación móvil.

8.7.1. Back office

Dentro del desarrollo de aplicaciones web hay una tendencia importante hacia las denominadas SPA: *Single Page Applications* [32]. Uno de los principales objetivos es conseguir una mejora importante en la experiencia de usuario: la mejora de los tiempos de espera o latencia entre vistas (similar a las aplicaciones nativas).

Habitualmente la lógica de negocio (el código ejecutable) de aplicaciones web se realiza íntegramente en el lado del servidor, y se confía la propia naturaleza del sistema de vistas a la tecnología *front-end*. Para el navegador, cada URL diferente es completamente independiente del resto. A pesar de que tenga los mismos estilos y/o plantillas, estos tienen que volver a ser procesados desde cero. Esto, para la gran mayoría de páginas web dinámicas, implica que al cambiar entre vistas se sufrirá el problema de la latencia en la web.

Esta arquitectura implica que el estado de la aplicación del cliente es difícil de mantener, teniendo que hacer auténticos malabarismos para poder gestionar una simple transferencia de información de una vista a otra. Es por todo esto que buscaremos un sistema que nos proporcione un desarrollo SPA, acelerando así el proceso de implementación aprovechando las características de definición de componentes y mejoras de procesamiento que nos proporciona.

8.7.1.1. JavaScript

Es un lenguaje con muchas posibilidades, utilizado para crear pequeños programas que luego son insertados en una página web y en programas más grandes, orientados a objetos mucho más complejos. Con JavaScript [33] podemos crear diferentes efectos e interactuar con nuestros usuarios. Es un lenguaje de programación que surgió por la necesidad de ampliar las posibilidades del HTML. JavaScript como lenguaje base, ha ido desarrollando multitud de *frameworks* destinados a la mejora de determinados elementos web, o a la implementación de nuevos paradigmas de desarrollo como el SPA. Para este paradigma, encontramos algunos de mayor relevancia en el mercado profesional:

ReactJS

Es una biblioteca JavaScript de código abierto para crear interfaces de usuario con el objetivo de animar al desarrollo de aplicaciones en una sola página. Es mantenido por Facebook, Instagram y una comunidad de desarrolladores independientes y compañías. Mantiene un control virtual de elementos de la vista propio, en lugar de confiar solamente en el del navegador. Esto deja a la biblioteca determinar qué partes han cambiado comparando contenidos entre la versión nueva y la almacenada, y utilizando el resultado para determinar cómo actualizar eficientemente la vista del navegador. Adopta la idea de un flujo de datos unidireccional. La curva de aprendizaje es relativamente corta y ,aunque no contiene todos los componentes de un *framework front-end*, hay una gran comunidad de *open source* alrededor que ha creado bibliotecas para manejo de datos, routing y animaciones en ReactJS [34].

EmberJS

EmberJS [35] es un framework JavaScript de código abierto basado en la arquitectura modelo-vista-controlador (MVC). Está catalogado como unos de los principales *frameworks* en el mundo de JavaScript ya que permite a los desarrolladores crear aplicaciones de una sola página (single-page) escalables.

Trata de estandarizar el desarrollo de aplicaciones del lado del cliente, y para eso utiliza reglas muy estrictas para crear un potente y sencillo desarrollo de nuestras aplicaciones. Para cumplir este objetivo posee una serie de clases y procedimientos los cuales nos ayudan a *renderiza*r nuestras aplicaciones y controlar la información.

AngularJS

AngularJS [36] es un framework MVC de JavaScript que permite crear aplicaciones SPA. Permite extender el vocabulario HTML con directivas y atributos, manteniendo la semántica y sin necesidad de emplear librerías externas.

A diferencia de la mayoría de sistemas de 'templates' y/o frameworks, Angular usa un sistema en el que vista y modelo están en relación constante, se considera el modelo como 'Single-Source-of-Truth'. Gracias a esto, se logra que todo cambio visual se actualice a tiempo real en el modelo y viceversa, evitando que sea el desarrollador el encargado de lograr la sincronía entre modelo y vista, como es el caso de otros *frameworks*.

AngularJS es una tendencia actual, donde el desarrollo de nuevas funcionalidades es constante, y su funcionamiento lo hace idóneo para el desarrollo de aplicaciones web como la que nosotros vamos a desarrollar. Además, otro punto fuerte por el que decantarnos por esta opción es la compatibilidad con tecnologías que nos permitirán migrar nuestra aplicación a multiplataforma una vez finalizado el desarrollo.

8.7.2. Aplicación móvil

La imparable expansión de los teléfonos inteligentes ha generado un mapa de oportunidades sin precedentes para el mercado de las aplicaciones móviles. Esta situación ha supuesto todo un revulsivo para desarrolladores y empresas que han visto en este campo un nicho de negocio para crear aplicaciones, gratuitas o de pago, de éxito, con muy pocos recursos.

En un principio la creación de las aplicaciones móviles se basa únicamente en el desarrollo de la aplicación en un lenguaje específico que varía en función de la plataforma a la que esté destinada nuestra aplicación (ya sea iPhone, Android, Windows…). Conocidas como aplicaciones nativas, son desarrolladas en código puro y hace necesario un conocimiento amplio sobre las tecnologías a utilizar. Además, conlleva contar con un equipo de expertos que puedan abarcar cada uno de los lenguajes sobre los que desarrollar para cada una de las plataformas.

Existen ya en el mercado varios servicios que permiten crear desde cero y paso a paso aplicaciones para las diferentes plataformas móviles, utilizando los asistentes de edición para personalizar el diseño de la interfaz y configurar las funciones de aplicaciones básicas. Una vez terminada nuestra creación, los mismos asistentes se encargan de compilar nuestro proyecto en una aplicación nativa para la plataforma elegida, guiándonos finalmente en el proceso de publicación de la misma en la tienda de aplicaciones.

8.7.2.1. Aplicaciones nativas

Una aplicación nativa es una aplicación que se desarrolla directamente en el lenguaje nativo de cada terminal. Por eso tendremos que utilizar un lenguaje diferente para cada Sistema Operativo. Los lenguajes de programación serán por tanto los siguientes:

- iOS: Objective C
- Android: Java
- Windows: C# y Visual Basic .NET.
- BlackBerry 10: C++

Una aplicación nativa es la opción cuyo resultado es el más robusto y fluido ya que se desarrolla directamente para integrarse en el Sistema Operativo. Si surge de una buena idea y un diseño bien trabajado a todos los niveles, la experiencia de usuario será completa ya que su funcionamiento, rendimiento y respuesta será el más inmediato de todas las opciones de desarrollo, incluso en los diseños más complejos y personalizados.

- Acceso a todo el hardware del móvil como GPS, cámara y demás accesorios.
- Acceso a todas las librerías gráficas del SO.
- Envío de notificaciones.
- Sincronizar o cachear datos para funcionar sin conexión a internet.
- Estar en las diferentes *stores* de aplicaciones.

Las desventajas son fundamentalmente del tipo económico ya que, como decíamos antes, para desarrollar aplicaciones nativas debemos conocer los diferentes lenguajes de

programación de cada Sistema Operativo. No será posible reutilizar el código de un SO en otro. Y no solo eso, debemos tener en cuenta que las Apps necesitan actualizaciones a nuevas versiones del SO, mantenimiento y/o aumento y mejoras de las funcionalidades… y todas estas cosas deben hacerse directamente en el código nativo de cada plataforma. Esto implica una alta inversión de tiempo para la adquisición de los conocimientos, y para el desarrollo y mantenimiento de cada una de las aplicaciones.

8.7.2.2. Aplicaciones web

Se trata de una web a la que se accede a través de una URL en el navegador del dispositivo (Safari, Chrome…) y se adapta al formato de tu pantalla para que tenga aspecto de navegación App. Los navegadores de los móviles permiten crear un acceso directo en nuestro escritorio de esta web, así esa será la manera de "instalarla" en nuestro dispositivo.

A nivel de lenguajes de programación, al ser una web deberemos usar lenguajes de programación web (HTML, CSS y Javascript).

8.7.2.3. Aplicaciones híbridas

Generalmente consisten en aplicaciones que contienen en su interior el navegador web del dispositivo. Para su desarrollo se utilizan *frameworks* de desarrollo basados en lenguajes de programación web (HTML, CSS y JS).

En este tipo de aplicaciones el nivel de integración con el SO dependerá del *framework* de desarrollo utilizado y cómo de abierto sea el SO, teniendo cada uno de ellos sus ventajas e inconvenientes. Actualmente con esta opción se obtiene acceso casi completo al hardware del teléfono e incluso en algunos casos a las librerías del SO, pero de momento no se ha conseguido igualar la respuesta y la experiencia de usuario de una App nativa.

Su uso es una opción muy económica y muy interesante para llegar al mayor número de usuarios repartidos en las diferentes plataformas y dispositivos aunque por el momento sus limitaciones son claras.

PhoneGap

PhoneGap [37] es un paquete de librerías que permite empaquetar aplicaciones HTML5 de manera que puedan ser usadas como aplicaciones para móviles o web. Es una solución de Adobe que nos permite llevar el desarrollo para la web al mundo de los dispositivos. Nos permite ejecutar aplicaciones desarrolladas con HTML, CSS y Javascript como si fueran aplicaciones nativas para los teléfonos móviles o tablets.

Las aplicaciones que podemos desarrollar con PhoneGap se pueden publicar en las conocidas tiendas de aplicaciones (Google Play, Windows Store o App Store de Apple) y, al igual que las aplicaciones nativas, también son capaces de acceder a los periféricos de los dispositivos como la cámara, acelerómetro, etc.

PhoneGap contiene una serie de librerías que te facilitan todas las utilidades que puedes llegar a necesitar sobre un teléfono. En lugar de aprender las librerías propias de cada sistema con sus lenguajes, haces uso de aquellas que te proporciona el *framework*, usando un único lenguaje de programación, JavaScript.

Ionic

Ionic [38] es un *framework* gratuito y *open source* para desarrollar aplicaciones híbridas multiplataforma que utiliza HTML5, CSS (generado por SASS) y Apache Cordova como base. Es uno de los *framework* del momento por utilizar AngularJS para gestionar las aplicaciones, lo que asegura aplicaciones rápidas y escalables.

Utiliza Apache Cordova para proporcionarnos la estructura de aplicación mínima sobra la que poder comenzar a trabajar, e Ionic en sí nos ofrecerá facilidades en el desarrollo de la interfaz de usuario. Con esta dupla, AngularJS con su versatilidad y potencia para la creación de aplicaciones e Ionic Framework para el desarrollo de la interfaz, obtenemos una herramienta de creación de aplicaciones completísima, con la que ahorraremos tiempo y trabajo en el desarrollo de cada proyecto. Otras características que incorpora para el desarrollo de aplicaciones son el reconocimiento táctil, lógica de animación de interfaces, verificador HTML o comunicación asíncrona.

Con esta premisa obtenemos un método que nos permitirá desarrollar nuestro proyecto para una versión de aplicación web que, una vez finalizado, podrá ser escalable a entorno multiplataforma a través del uso de la plataforma Ionic sin la necesidad de desarrollar cada aplicación de forma nativa y partiendo desde cero.

8.8. Diseño del control de versiones

Un sistema de control de versiones es una herramienta que registra todos los cambios hechos en uno o más proyectos, guardando así versiones del producto en todas sus fases del desarrollo. Las versiones son como fotografías que registran su estado en ese momento del tiempo y se van guardando a medida que se hacen modificaciones al código fuente.

Antes de la masificación de los sistemas de control de versiones, se solían guardar las iteraciones del proyecto en archivos comprimidos y medios de almacenamiento como diskettes y CD. Los sistemas de control de versiones nacen de la necesidad de solventar y facilitar este tedioso proceso.

Para un desarrollador esta herramienta es muy valiosa porque permite viajar atrás en el tiempo (hacer *roll-back*) si los cambios aplicados no resultaron de la manera que se esperaba, pudiendo restaurar en cualquier momento una versión previa. Es como un respaldo permanente. Hoy en día son usados no solo por desarrolladores independientes sino también por *Startups* y grandes corporaciones; “La posibilidad de volver atrás no tiene precio”.

8.8.1. Subversion

Subversion [39] es un sistema de control de versiones libre y de código fuente abierto. Crea un árbol de ficheros en un repositorio central, como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos.

Puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones, y, puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único—si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio.

8.8.2. Mercurial

Mercurial [40] es un sistema de control de versiones distribuido libre y gratuito. Rompe con el modelo tradicional de trabajo cliente/servidor de los repositorios como Subversión.

En los repositorios tradicionales existe una clara diferencia entre los conceptos servidor y cliente: el servidor es el que mantiene y controla el versionado de los ficheros, y el cliente se descarga copias del servidor para que sean modificadas y subidas posteriormente con el fin de que sean vistas por el resto de usuarios. En cambio, en los repositorios distribuidos, los clientes ya no se descargan una copia del repositorio, sino que lo clonan, comportándose como cliente y servidor al mismo tiempo, es decir, que la copia podría comportarse a su vez como servidor ante otros clientes (construyendo así una estructura distribuida).

8.8.3. Git

Git [41] es un software de control de versiones pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. ¿Qué nos aporta?

- Auditoría del código: saber quién ha tocado qué y cuándo
- Control sobre cómo ha cambiado nuestro proyecto con el paso del tiempo
- Control de versiones a través de etiquetas: versión 1.0, versión 1.0.1, versión 1.1, etc. Sabremos exactamente que había en cada una de ellas y las diferencias entre cualquiera de ellas dos
- Seguridad: todas las estructuras internas de datos están firmadas con SHA1. No se puede cambiar el código sin que nos enteremos
- Merging y branching extremadamente eficientes

Para el uso del software Git será necesaria una herramienta que nos dé acceso y control sobre ella. Aquí es donde entra en juego GitLab [42], un proyecto de código libre que se puede instalar en tu propio servidor y que te permite tener repositorios privados o colaborativos sin coste alguno: En nuestro caso trabajaremos con un repositorio privado. El uso de la herramienta nos permitirá tener todo el servicio bajo control, pudiendo instalarlo en cualquier plataforma UNIX/Linux.

El uso de Git nos va a permitir trabajar perfectamente con la metodología de desarrollo planteada. Para llevarla a cabo en primer lugar crearemos el repositorio en la página de GitLab. Una vez tengamos el proyecto creado nos proporcionará unas claves que nos permitirán conectarnos al repositorio del proyecto y poder interactuar entre los diferentes clientes. Por último, configuraremos los repositorios tanto en local (para el desarrollo) como en el servidor (para el despliegue) con el uso de las claves.

	SVN	Git	Mercurial
Control de versiones	Centralizada	Distribuida	Distribuida
Repositorio	Un repositorio central donde se generan copias de trabajo	Copias locales del repositorio en las que se trabaja directamente	Copias locales del repositorio en las que se trabaja directamente
Autorización de acceso	Dependiendo de la ruta de acceso	Para la totalidad del directorio	Para la totalidad del directorio
Seguimiento de cambios	Basado en archivos	Basado en contenido	Numeración consecutiva auxiliar
Historial de cambios	Solo en el repositorio completo	Tanto en el repositorio como en las copias de trabajo	Tanto en el repositorio como en las copias de trabajo
Conectividad de red	Con cada acceso	Solo necesario para la sincronización	Solo necesario para la sincronización

Tabla 4. Herramientas de control de versiones
(Fuente propia)

8.9. Diseño Interfaces

Para el diseño de interfaces optaremos por el desarrollo de *mock-ups*. Este modelo busca el diseño del resultado final, y, aunque requiere invertir un mayor tiempo en el proceso de diseño, nos agilizará la implementación de las vistas dado que conoceremos con exactitud cuáles serán los elementos y las funcionalidades que aplicaremos sobre cada una de ellos, junto con todos sus características de diseño: colores, posición en la ventana, tamaño de fuente, etc.

8.9.1. Aplicación móvil

- [RF-USR-01]: Registro – Cualquier usuario podrá registrarse en la aplicación.

Figura 15. Mock-up aplicación móvil: Registro
(Fuente propia)

- [RF-USR-02]: Iniciar sesión – Un usuario registrado podrá acceder a la aplicación.

Figura 16. Mock-up aplicación móvil: Iniciar sesión
(Fuente propia)

- [RF-USR-03]: Cerrar sesión – Un usuario registrado podrá cerrar su sesión en la aplicación. Operación disponible en el menú de ajustes.

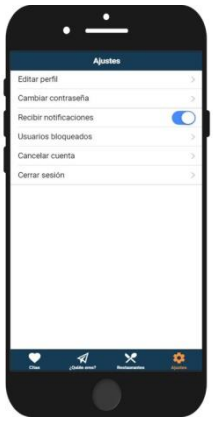

Figura 17. Mock-up aplicación móvil: Cerrar sesión
(Fuente propia)

- [RF-USR-04]: Formulario - Un usuario registrado debe completar el formulario de personalidad para poder disfrutar de sus funcionalidades. Una vez completado podrá ser modificado.

Figura 18. Mock-up aplicación móvil: Formulario
(Fuente propia)

- [RF-USR-05]: Generar cita - Un usuario puede generar citas. Podrá seleccionar fecha y restaurante en base al lugar de la cita.

Figura 19. Mock-up aplicación móvil: Generar cita
(Fuente propia)

- [RF-USR-06]: Aceptar cita - Un usuario que ha sido elegido por compatibilidad para una cita podrá aceptar una cita.

Figura 20. Mock-up aplicación móvil: Aceptar cita
(Fuente propia)

- [RF-USR-07]: Modificar cita - Un usuario podrá modificar los datos de una cita aún no aceptada. Opción disponible en los detalles de la cita.

Figura 21. Mock-up aplicación móvil: Modificar cita
(Fuente propia)

- [RF-USR-08]: Listar citas - El usuario tendrá acceso a un listado de las citas que ha generado y en las que ha sido elegido.

Figura 22. Mock-up aplicación móvil: Listar citas
(Fuente propia)

- [RF-USR-09]: Cancelar cita – Un usuario registrado podrá cancelar una cita. Operación disponible en los detalles de la cita.

Figura 23. Mock-up aplicación móvil: Cancelar cita
(Fuente propia)

- [RF-USR-10]: Valorar cita - Un usuario podrá valorar una cita una vez se haya realizado.

Figura 24. Mock-up aplicación móvil: Valorar cita
(Fuente propia)

- [RF-USR-11]: Listar restaurantes - Un usuario podrá listar los restaurantes existentes en la aplicación.

Figura 25. Mock-up aplicación móvil: Listar restaurantes
(Fuente propia)

- [RF-USR-12]: Cancelar cuenta - Cualquier usuario registrado podrá cancelar su cuenta. Operación disponible en el menú de ajustes.

Figura 26. Mock-up aplicación móvil: Cancelar cuenta
(Fuente propia)

- [RF-USR-13]: Modificar datos de la cuenta - Cualquier usuario registrado podrá modificar los datos relativos a la cuenta.

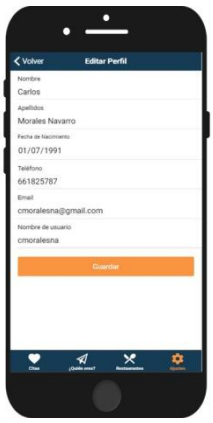

Figura 27. Mock-up aplicación móvil: Modificar datos de la cuenta
(Fuente propia)

- [RF-USR-14]: Cambiar contraseña - Cualquier usuario registrado podrá modificar la contraseña para el acceso de su usuario al sistema.

Figura 28. Mock-up aplicación móvil: Cambiar contraseña
(Fuente propia)

- [RF-USR-15]: Gestionar usuarios bloqueados - Cualquier usuario registrado podrá gestionar los bloqueos hacia posibles candidatos en las citas generadas.

Figura 29. Mock-up aplicación móvil: Gestionar usuarios bloqueados
(Fuente propia)

8.9.2. Back office

- [RF-RST-01]: Registro - Un usuario designado por el restaurante (en adelante, encargado) podrá registrar a éste en la aplicación.

Figura 30. Mock-up aplicación web: Suscripción
(Fuente propia)

- [RF-RST-02]: Iniciar sesión - Un encargado registrado podrá acceder al sistema

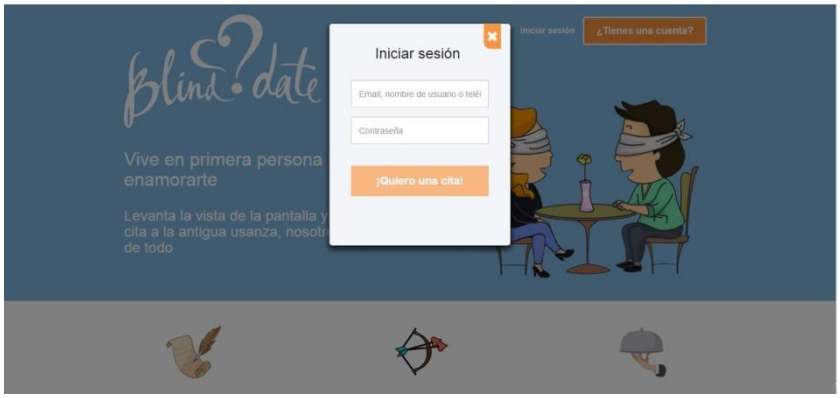

Figura 31. Mock-up aplicación web: Iniciar sesión
(Fuente propia)

- [RF-RST-03]: Gestionar fotografía - Un encargado gestionará la fotografía referente al restaurante. Esta opción estará disponible en el menú lateral.

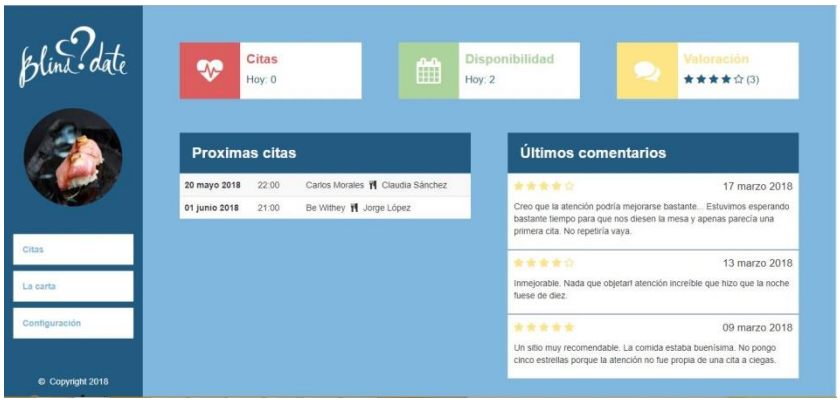

Figura 32. Mock-up aplicación web: Gestionar fotografía
(Fuente propia)

- [RF-RST-04]: Gestionar publicación - El encargado será capaz de crear, modificar y eliminar en su tarjeta personal la publicación referida a su oferta y menú.

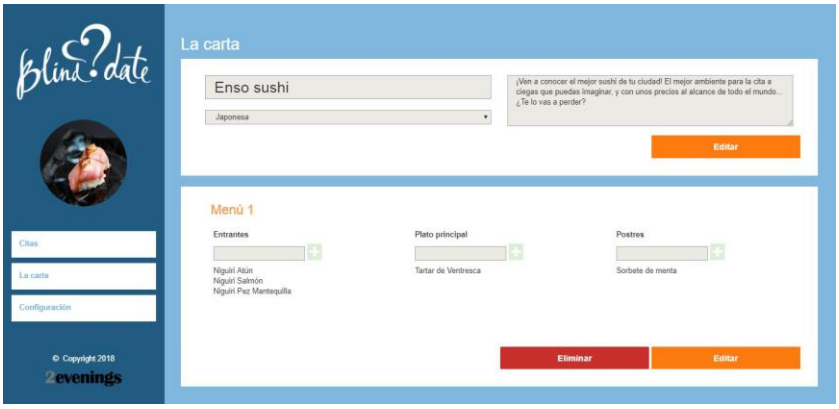

Figura 33. Mock-up aplicación web: Gestionar publicación
(Fuente propia)

- [RF-RST-05]: Listar citas - Un encargado podrá listar las citas relativas a su restaurante.

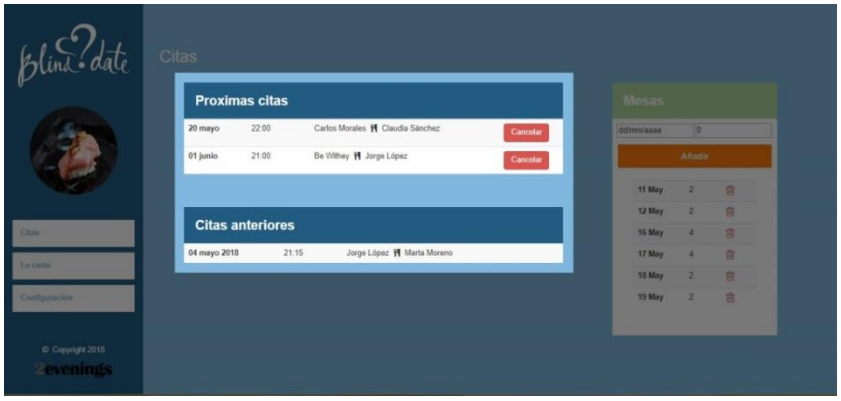

Figura 34. Mock-up aplicación web: Listar citas
(Fuente propia)

- [RF-RST-06]: Cancelar citas - Un encargado podrá cancelar una cita relativa a su restaurante.

Figura 35. Mock-up aplicación web: Cancelar citas
(Fuente propia)

- [RF-RST-08]: Modificar disponibilidad - Un encargado podrá modificar la disponibilidad del restaurante para celebrar citas.

Figura 36. Mock-up aplicación web: Modificar disponibilidad
(Fuente propia)

- [RF-RST-09]: Modificar datos de la cuenta - El encargado podrá modificar la suscripción del restaurante.

Figura 37. Mock-up aplicación web: Modificar datos de la cuenta
(Fuente propia)

- [RF-RST-10]: Carmbiar contraseña - El encargado podrá modificar la contraseña para el acceso al sistema.

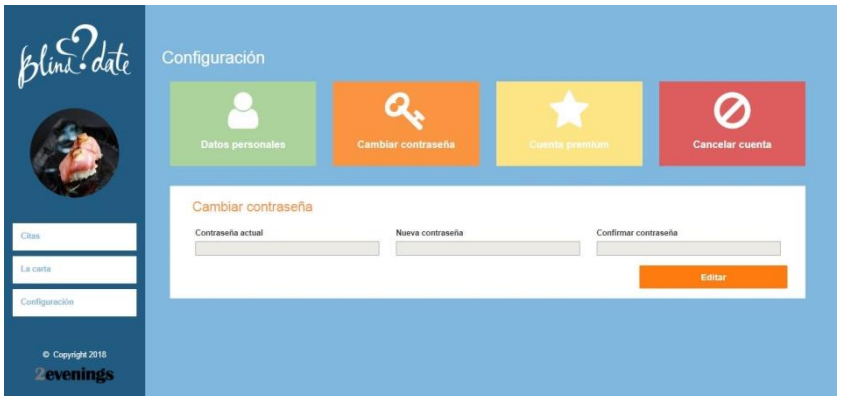

Figura 38. Mock-up aplicación web: Cambiar contraseña
(Fuente propia)

- [RF-RST-11]: Cancelar suscripción - El usuario podrá cancelar la suscripción del restaurante.

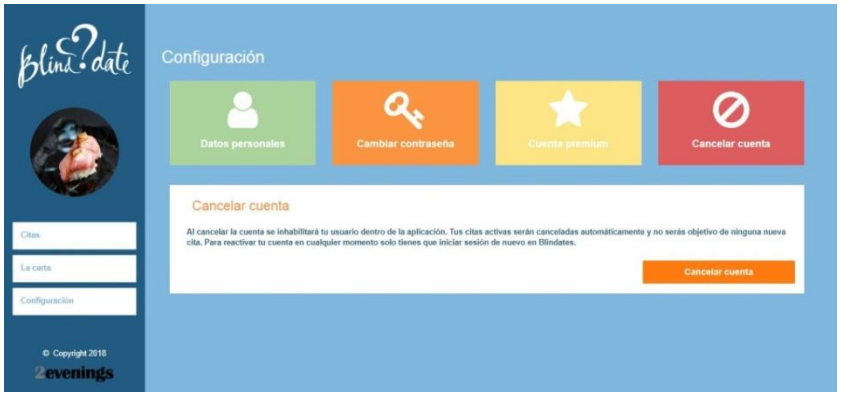

Figura 39. Mock-up aplicación web: Cancelar suscripción
(Fuente propia)

8.10. Diseño de pruebas y validación

Las pruebas de integración y de correcto funcionamiento son algo imprescindible en el desarrollo de cualquier proyecto. Es por ello que en la mayoría de las compañías podemos encontrar un equipo dedicado exclusivamente al desarrollo y ejecución de diferentes pruebas que nos aseguren que el producto está listo para ponerlo en funcionamiento en el entono público.

En nuestro caso, dada la limitación de tiempo de la que disponemos, se nos hace imposible invertir el tiempo suficiente y deseado en la realización de las pruebas. Ideas como la automatización de casos de prueba o los test de integración quedan muy lejos de los tiempos con los que nos movemos. Pero, aun así será necesario definir determinados casos que nos aseguren de forma general que nuestro desarrollo funciona.

Si volvemos al esquema temporal que hemos definido anteriormente en la *Tabla 2*, podemos observar que durante cada ciclo invertiremos un tiempo en la ejecución de las pruebas relacionadas con los desarrollos realizados durante el ciclo anterior.

El proceso para la creación de pruebas será definido durante la creación de los elementos de trabajo para cada ciclo. Al principio del ciclo, agruparemos los diferentes desarrollos sobre los que vamos a trabajar, con la idea de identificar todos los posibles detalles para su implementación, creando así las especificaciones de cada elemento. Una vez las especificaciones han sido definidas, crearemos los casos de prueba que consideremos imprescindibles para cada uno de los elementos, tanto para el *front-end* como para el *back-end*.

Las pruebas a realizar se harán de forma manual, siguiendo los pasos que hayamos definido para cada uno de los casos. En caso de que alguna de las pruebas realizadas no sea satisfactoria, crearemos una nueva entrada de tipo *bug* en nuestro sistema de gestión de trabajo, TFS, y ésta será añadida como elemento trabajo para el siguiente *sprint*. El *bug* hará referencia internamente a la entrada original de desarrollo fallida, con el fin de agilizar la búsqueda de las especificaciones definidas y casos de prueba.

En el momento de comprobar y validar el funcionamiento de un *bug*, seguiremos de nuevo los casos de prueba que definimos originalmente en la fase de análisis y toma de especificaciones.

9. Implementación

9.1. Visión general

Nuestra implementación comienza con la primera de las sesiones: identificación de los diferentes elementos sobre los que tendremos que trabajar a lo largo de nuestro proyecto. Esta sesión tratará de listar cada uno de los elementos, de forma general, sobre los que será necesario trabajar en el proyecto. La metodología SCRUM, por definición, tratará cada uno de los *sprints* como si de un proyecto individual se tratase, y planificará iterativamente cada uno de ellos. Con esta sesión no pretendemos planificar el proyecto completo, simplemente buscamos una visión de cada bloque a desarrollar. Esta lista de elementos irá variando a lo largo de la implementación del proyecto, ya que la metodología ágil nos permitirá identificar nuevos elementos a añadir o elementos que debamos eliminar de nuestro desarrollo.

Como hemos mencionado anteriormente, para la gestión de los elementos de trabajo dentro de los ciclos de desarrollo, utilizaremos la herramienta de Team Foundation Server. Esta herramienta nos permitirá definir los elementos de trabajo que deseemos, quedando disponibles para su futura asignación al *sprint* que corresponda. En el Apéndice I podemos observar la lista de elementos que definimos a partir de esta primera sesión. Durante las sesiones de análisis y estimaciones, moveremos cada uno de los elementos de trabajo al correspondiente ciclo de desarrollo, y, una vez allí, trabajaremos sobre los requerimientos y los distintos sub-elementos que cada uno de ellos engloba.

En adelante analizaremos el trabajo realizado en cada uno de los sprints, enfocándonos en los puntos más importantes y los resultados obtenidos al final de cada uno de los ciclos de desarrollo, siguiendo el esquema de proceso reflejado anteriormente en la *Tabla 2.*

Dividiremos cada uno de los *sprints* en tres bloques: Análisis y Estimaciones, Desarrollo, y Pruebas y Validación. Durante la fase de análisis y estimaciones expondremos los diferentes elementos de trabajo que serán desarrollados durante el ciclo. Añadiremos en cada uno de los elementos las sub-tareas correspondientes para completarlo, y, seguidamente, daremos un peso de tiempo de desarrollo a cada una de las sub-tareas con el fin de asegurarnos que somos capaces de cumplir nuestra meta al final del *sprint (n*ótese que en proyectos dónde el límite de tiempo no es un factor determinante, los tiempos no serán tan ajustados y la no realización de alguna de las tareas, de forma justificada, no generará un impacto tan grande como en nuestro caso). A continuación procederemos con el desarrollo de cada una de las sub-tareas, identificando los posibles problemas que puedan surgir y las soluciones aplicadas para que no bloqueen nuestro progreso. Por último, en la fase del *testing* analizaremos los casos de prueba que han sido aplicados a los elementos y el resultado obtenido en cada uno de ellos. Además, incluiremos los posibles errores encontrados en la lista de elementos sobre los que trabajar en futuros *sprints.*

9.2. Sprint 1

9.2.1. Análisis y Estimaciones

La primera fase del *sprint* consistirá en la identificación de los elementos a desarrollar durante el ciclo. En primer lugar crearemos en TFS la nueva iteración de trabajo y la incluiremos en nuestro proyecto. Esta funcionalidad la encontraremos dentro del menú de la herramienta, en la sección de trabajo. Encontraremos toda la información necesaria sobre los pasos a seguir en la documentación de Microsoft [43].

La Figura 40 nos muestra el resultado una vez hemos creado la iteración dentro de nuestro proyecto. Será posible asignar ahora los elementos de trabajo deseados a ésta.

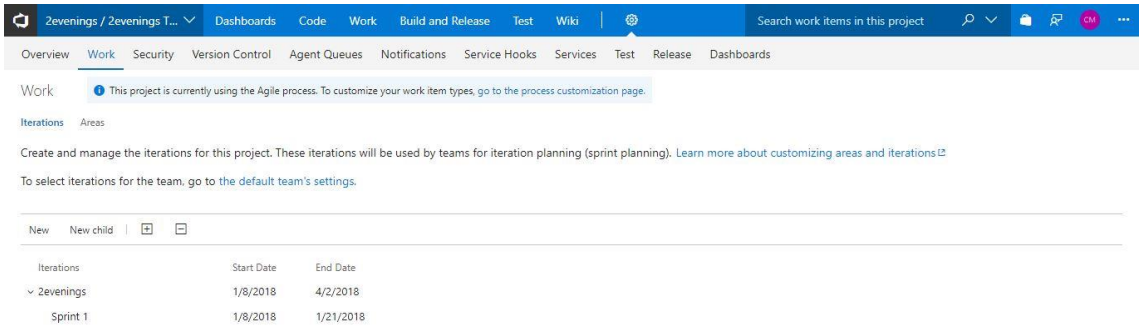

Figura 40. Sprint 1: Creación de la iteración en TFS
(Fuente propia)

Para designar los elementos que compondrán nuestro *sprint* nos basaremos en establecer un objetivo general de resultado. Al final de cada *sprint* es esperado que el resultado del desarrollo componga de forma independiente un proyecto totalmente funcional. Por lo tanto, los elementos que incluyamos para el ciclo de desarrollo deberán asegurarnos que, una vez éste haya concluido, seremos capaces de generar una nueva versión del proyecto que pueda ser validada durante la fase de pruebas. Así pues, al comienzo de cada fase de análisis buscaremos definir nuestro resultado u objetivo esperado. Para este primer *sprint*, nuestro objetivo será: "Configuración del servidor y primeras funcionalidades del *back-office*".

Una vez definido el objetivo, debemos identificar, de entre los elementos que definimos en primera instancia durante la visión general del proyecto, disponibles en el Apéndice I, cuáles de ellos son los esenciales para el resultado esperado. Estos elementos se encuentran en primera instancia en nuestra reserva, y ahora serán incluidos en la nueva iteración para ser analizados y proporcionar una estimación para cada uno de ellos.

La Figura 41 nos muestra el panel de nuestra iteración una vez que los elementos han sido incluidos en esta. Como podemos observar, los elementos incluidos se componen de los relativos a la adquisición y gestión del dominio, adquisición y configuración del servidor para la creación de los *sites*, y la creación de la *landing page*, *login* y registro dentro de nuestra aplicación, buscando así nuestra primera versión funcional de la aplicación.

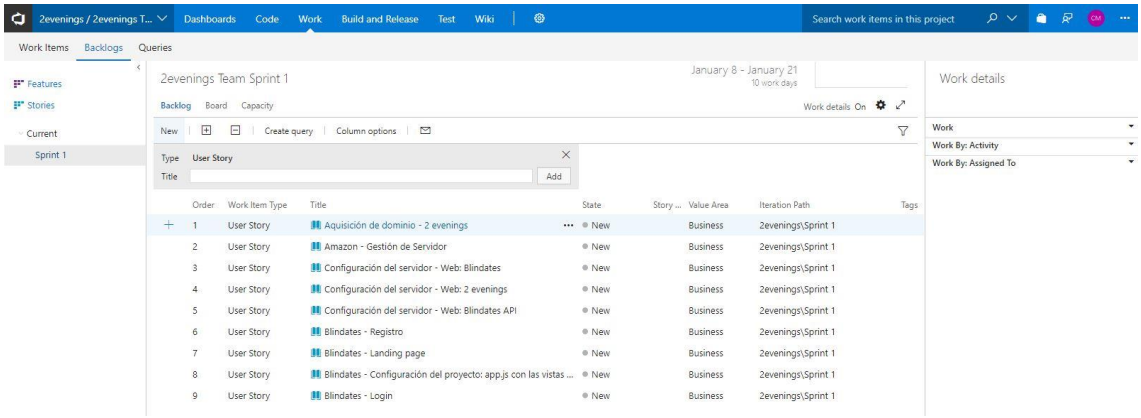

Figura 41. Sprint 1: Elementos añadidos a la iteración
(Fuente propia)

Es el momento de analizar en detalle cada uno de los elementos de la lista. El objetivo de este análisis es definir con detalle qué esperamos obtener al completar cada una de las tareas y cómo vamos a proceder en el desarrollo. En este punto, habiendo realizado de antemano la toma de decisiones en cuanto a tecnologías y habiendo definido cada uno de los elementos de los que se compondrá la vista a partir de los *mock-ups*, deberemos detallar los sub-objetivos que comprende cada tarea, o, lo que es lo mismo, el desarrollo implícito que supone cada una de las tecnologías para completar la tarea.

Para ello, TFS de nuevo nos proporciona una herramienta de creación de tareas para cada *Story*. *Story*, en adelante, será el nombre con el que nos refiramos a ese elemento global de desarrollo que comprenderá una funcionalidad en nuestro sistema. Por tanto, a lo largo del análisis, identificaremos esas sub-tareas y las enlazaremos a la funcionalidad que corresponda.

Una vez un *Story* contenga todos los puntos que consideramos necesarios para su desarrollo, procederemos a la estimación del tiempo que requiere completar la tarea. Las estimaciones en la metodología SCRUM se realizan en lo que conocemos como *planning póker*. Es una sesión que tiene lugar al comienzo de cada *sprint*, en la que se reúnen todos los integrantes del equipo, habiendo analizado con anterioridad todos los *Stories* que compondrán el *sprint*, y, en ella, expondrán al equipo la propuesta de proceso con el fin de buscar un acuerdo común para encontrar la mejor solución. A continuación, mediante el uso de cartas (de ahí el término *póker*) se procederá a la estimación individual del tiempo que requerirá la tarea. En caso de que haya diversidad de opiniones, se buscará el consenso común para proporcionar una única estimación. En el artículo sobre “La estimación ágil con la técnica Planning Póker” [44], Samuel Casanova, experto Scrum Master en Agilar, nos describe las características, elementos y dinámica de la sesión.

Al finalizar esta sesión, el resultado que obtenemos (disponible en el Apéndice II) es el de una lista de elementos de trabajo, con las diferentes características de desarrollo que cada uno de ellos engloba y el tiempo previsto para completarlo. Esto nos permitirá ahora planificar el tiempo desarrollo a lo largo de los días que hemos establecido.

9.2.2. Desarrollo

9.2.2.1. Configuración del servidor

La primera fase del desarrollo del *sprint* la dedicaremos a la configuración del servidor. Nuestro objetivo es el de tener en funcionamiento los diferentes dominios que especificábamos en el apartado de diseño, y que nuestro servidor esté configurado para resolver las llamadas a esos dominios y mostrar el contenido. Es decir, necesitamos gestionar la compra del dominio "2evenings.es" en una de las plataformas disponibles en el mercado, gestionar la creación de un servidor en "Amazon Cloud Services" y que cuando en cualquier explorador escribamos uno de los dominios mencionados, se muestre la página web correspondiente en nuestro servidor.

Para la compra del dominio, la elección de la plataforma no será merecedora de estudio. Simplemente basaremos la elección en el precio/tiempo que nos ofrezca la plataforma por la compra del dominio. Hostalia es una plataforma muy asequible, que nos proporciona el dominio deseado por un precio de 5€/año. Además, como mostramos en la Figura 42, Hostalia nos ofrece una plataforma de gestión de diferentes dominios, e incluso puede proporcionarnos un servidor configurado para alojar el dominio si lo deseamos, pero, como ya hemos mencionado, únicamente buscamos la gestión de la compra del dominio.

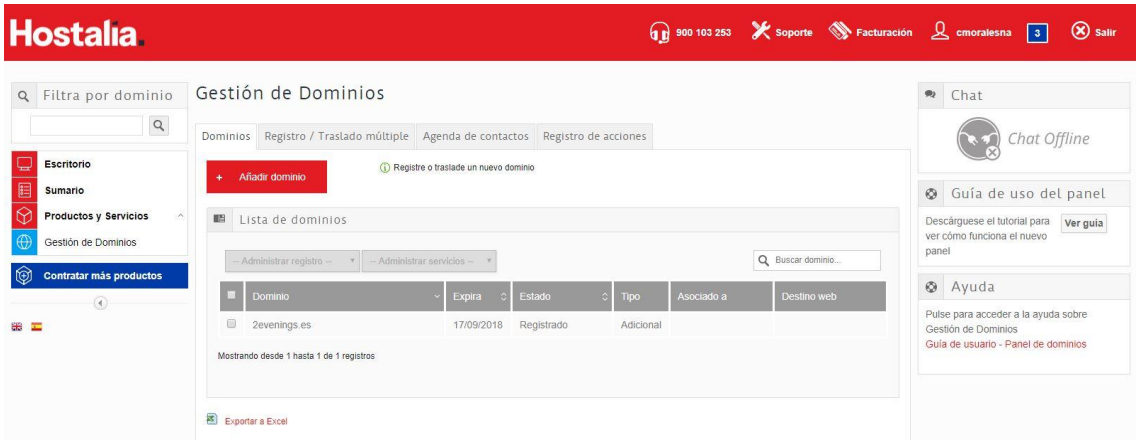

Figura 42. Hostalia: Panel de gestión
(Fuente propia)

Una vez tenemos en nuestro poder el dominio, el siguiente paso será el de la obtención y configuración de un servidor. Amazon nos ofrece una guía de los pasos necesarios para la creación y configuración de una instancia dentro de su sistema [45]. Siguiendo todos los pasos indicados, finalmente obtenemos el servidor con la configuración básica detallada durante el proceso de creación. Esta configuración inicial incluirá las claves necesarias para que podamos acceder al servidor, y la dirección IP de éste, mediante la cual podremos mostrar el contenido del servidor web en el explorador.

El primer problema con el que nos encontramos es que esta IP varía cada vez que la instancia se reinicia, y esto afectará en el momento de la configuración del dominio en el servidor. Será necesario entonces configurar una IP elástica, que nos permitirá que la IP pública (IP de acceso a nuestro servidor web) sea fija aunque la IP interna se modifique. Amazon nos explica

nuevamente los conceptos y los pasos para la configuración de la IP elástica en nuestra instancia en su guía [46], y además nos la ofrece de forma gratuita mientras se encuentre vinculada a una instancia. En la Figura 43 podemos observar el resultado de la configuración de la IP elástica sobre nuestra instancia.

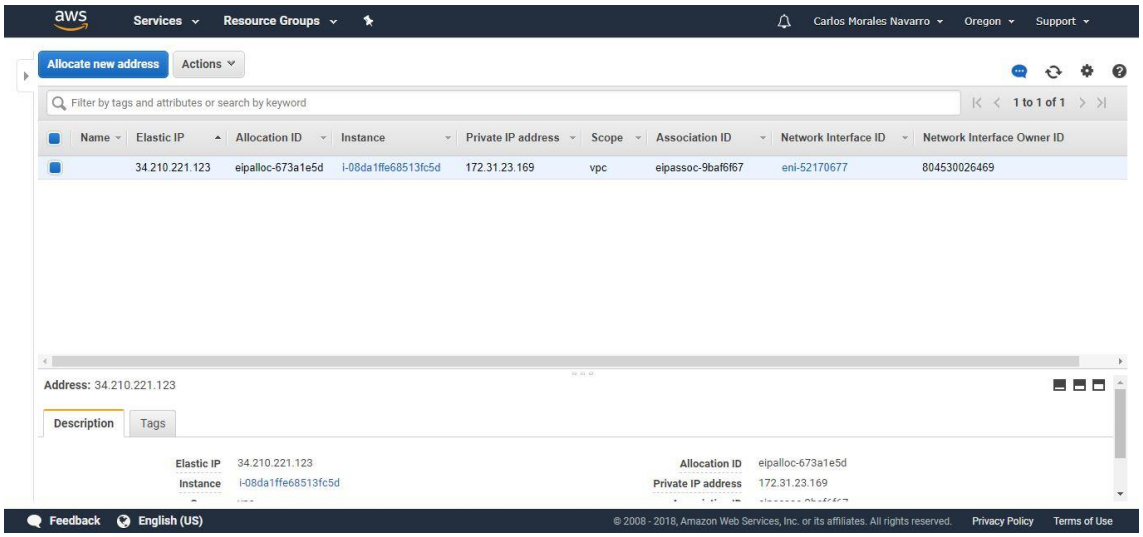

Figura 43. Amazon: Panel de gestión de IP elástica
(Fuente propia)

La IP elástica nos permitirá ahora trabajar con otra de las herramientas que nos proporciona Amazon Cloud Services: Route 53. Se trata de un servidor DNS en el que podremos configurar los diferentes dominios y subdominios que queremos alojar en nuestro sistema, indicándole en qué servidor se resolverán. Route 53 será el encargado de identificar el momento en el que un usuario acceda a alguna de nuestras urls, y servir la información de la web que hayamos configurado para el dominio. La IP elástica es necesaria dado que en la configuración debemos indicar la dirección del servidor a la cual deberá pedir la información de la web. De nuevo Amazon nos guía a través de los pasos necesarios para realizar la configuración adecuada en su guía [47]. En la Figura 44 podemos observar el resultado de la configuración aplicada.

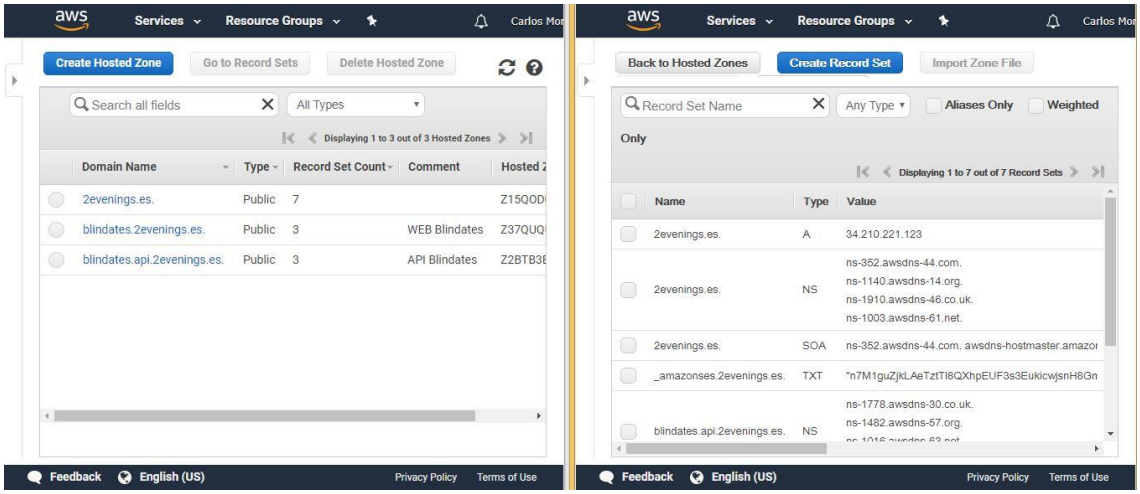

Figura 44. Amazon: Panel de configuración de Route 53
(Fuente propia)

Por último, para completar la configuración de nuestro servidor deberemos trabajar en la configuración de Apache, nuestro servidor web. En este momento, Route 53 se encuentra configurado para pedirle a nuestro servidor el contenido que debe mostrar para la url solicitada, pero debemos configurar nuestro servidor para que sea capaz de responder a la petición y proporcionarle la información necesaria.

Para ello, Apache define una serie de archivos en los que será necesario especificar los detalles de la consulta: de quién recibirá la petición y cuál es el directorio de archivos relativo a esa url. En la Figura 45 observamos la configuración realizada para la url 2evenings.es, donde hemos indicado en el parámetro "ServerName" cuál es la url de petición, y en "DocumentRoot" cuál es el directorio de los archivos de respuesta. Además, nos permite definir la dirección y nombre de los documentos que utilizaremos para la gestión de *logs*. Previamente, habremos definido dentro del directorio "home" el árbol de directorios que utilizaremos en nuestro servidor.

```
<VirtualHost *:80>
        ServerName 2evenings.es
        ServerAdmin webmaster@localhost
        DocumentRoot /home/web/2evenings.es/

        ErrorLog /home/log/2evenings.log
        CustomLog /home/log/access.2evenings.log combined

        <Directory />
            Options Indexes FollowSymLinks Includes ExecCGI
            AllowOverride All
            Require all granted
            Allow from all
        </Directory>

</VirtualHost>
```

Figura 45. Apache: Configuración de 2evenins.es
(Fuente propia)

9.2.2.2. Back-office

La segunda fase del *sprint* comprende lo relativo a la configuración y primeras vistas de nuestra aplicación de gestión. En la sección de diseño ya hemos hablado sobre AngularJS: es un *framework* de Javascript de código abierto para crear y mantener aplicaciones web de una sola página. En términos enfocados a la configuración del proyecto, el uso de AngularJS no supondrá nada más allá de indicarle a nuestra página web que cargue el código JavaScript que AngularJS define y necesita para su funcionamiento. Por tanto, procederemos como si de una web se tratase, con la creación del archivo principal (index.html) donde definiremos la inclusión de los archivos del *framework* junto con los archivos que nuestra solución requiera.

En esta primera iteración, deberemos definir los archivos necesarios para que nuestra *landing page* sea visible en el momento de la carga, y los componentes de *login* y registro para completar la primera versión funcional. La Figura 46 nos muestra los elementos que hemos definido para completar nuestras tareas del *sprint*. La primera de las etiquetas será la que permita a AngularJS trabajar sobre una única vista a lo largo de la aplicación.

La primera mitad de etiquetas se compone de inserción de diferentes *frameworks* y librerías que JavaScript nos proporciona y a las que daremos uso. La segunda mitad se compone de los diferentes elementos que hemos desarrollado, como componentes o servicios, para el funcionamiento de nuestra aplicación.

```
<div ui-view></div>

<!-- JQUERY -->
<script src="lib/jquery.min.js"></script>

<!-- BOOTSTRAP -->
<script src="lib/bootstrap.min.js"></script>

<!-- ANGULAR JS -->
<script src="lib/angular.min.js"></script>
<script src="lib/angular-resource.min.js"></script>
<script src="lib/angular-ui-router.min.js"></script>
<script src="lib/angular-sanitize.js"></script>
<script src="lib/angular-cookies.js"></script>
<script src="lib/ui-bootstrap.min.js"></script>

<!-- MODULES -->
<script src='app/access/unauthorised-module.js'></script>

<!-- CONFIGURATION -->
<script src='config/app.js'></script>
<script src="config/app.local.config.js"></script>

<!-- SERVICES -->
<script src='app/access/access-service.js'></script>
<script src='app/access/login-service.js'></script>
<script src='app/access/register-service.js'></script>

<!-- CONTROLLERS -->
<script src='app/access/login-controller.js'></script>
<script src='app/access/login-modal-controller.js'></script>
<script src='app/access/register-controller.js'></script>

<!-- CLASSES -->
<script src='Identity/User.js'></script>
<script src='Identity/UserPassword.js'></script>
<script src='Identity/GenericAddress.js'></script>
```

Figura 46. Sprint 1: Elementos definidos en el back-office (Fuente propia)

El principal archivo que necesitaremos definir para el funcionamiento de AngularJS será el de la configuración del módulo. AngularJS necesita de la existencia de un módulo principal sobre el que manejará las distintas inyecciones de dependencia sobre los archivos que hemos creado. Es decir, aunque nuestros archivos estén asociados al proyecto, será necesario configurar AngularJS para que sepa qué vistas existirán en nuestro sistema y qué archivos debe escoger para cada una de ellas.

El desarrollo de este *sprint* será comprendido en un único módulo, al que denominaremos “no autorizado”, ya que trabajamos con todas las vistas del sistema en las cuales el usuario no se encuentra autentificado. Sobre este módulo será necesario crear las diferentes vistas que lo compondrán: *landing page*, *login* y registro. Anteriormente, en el apartado de diseño, hemos podido ver el resultado del diseño de las dos últimas, pero será necesaria la definición del *mock-up* para la página de inicio antes de desarrollar ésta. La Figura 47 nos muestra el resultado del diseño. En ella podemos observar los elementos de acceso a nuestras otras dos vistas incluidas en el *sprint*.

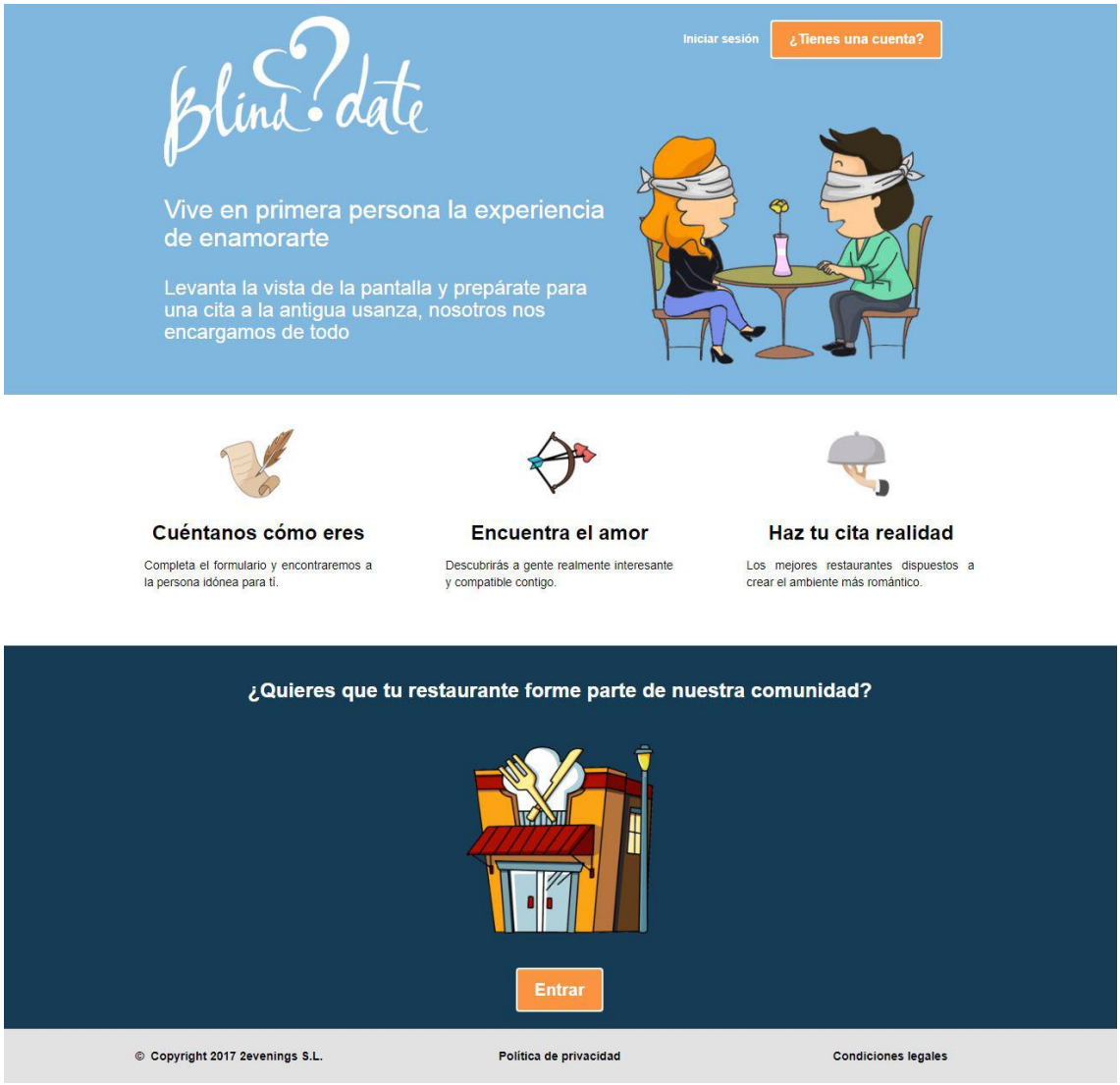

Figura 47. Mock-up aplicación web: landing page
(Fuente propia)

Por último, para finalizar nuestro desarrollo del *sprint*, procederemos con la creación de nuestra API Rest para completar las funcionalidades de *login* y registro, cuyas acciones ya hemos definido en la vista. Ahora Symfony será el encargado de recibir las llamadas que nuestro código en AngularJS realiza, y, procesando los datos, devolver la respuesta que se requiera para cada llamada.

Podremos encontrar la explicación y los pasos a seguir para la creación de la base de nuestro proyecto en el tutorial sobre Symfony 3 [48]. Al completarlo, obtendremos nuestro proyecto de Symfony creado satisfactoriamente. Uno de los requerimientos de la creación del proyecto será tener previamente definida la base de datos que nuestra API atacará, con el fin de crear los modelos dentro del proyecto siguiendo el paradigma de *entity-model* sobre el que se basa Symfony. Procederemos entonces con la instalación de la herramienta phpMyAdmin para la creación de la base de datos MySQL en nuestro servidor, y la definición de las tablas siguiendo el esquema anteriormente creado y reflejado en la Figura 12. La Figura 48 nos muestra el resultado de las tablas una vez creadas en nuestro sistema.

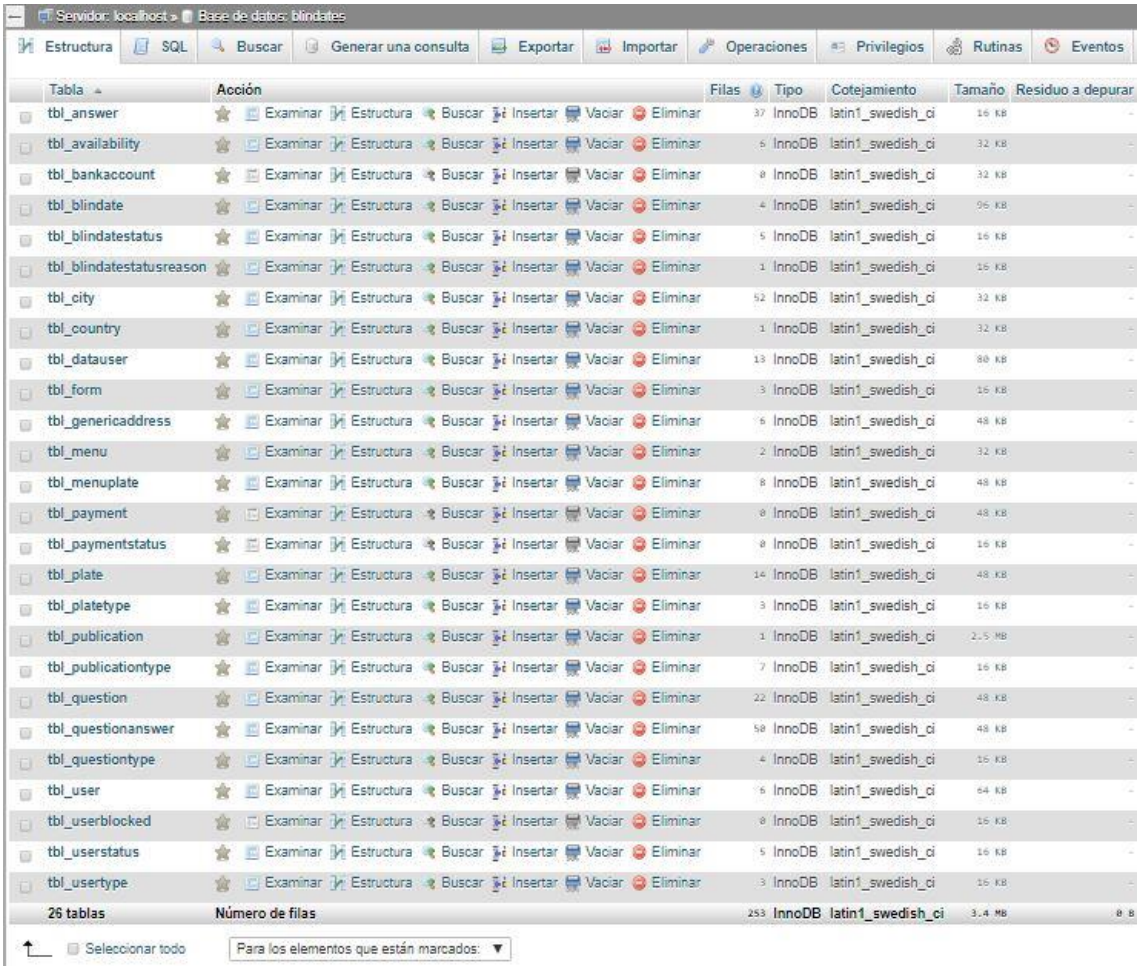

Figura 48. Sprint 1: Creación de las tablas en phpMyAdmin
(Fuente propia)

Configuraremos, por tanto, nuestro proyecto Symfony para que apunte a la nueva base de datos generada. Una vez nuestro proyecto ha sido lanzado, lo primero que podremos observar en él es la lista de entidades generadas, siguiendo como patrón el diseño que tenemos de las tablas en base de datos. Cada una de las entidades estará identificada con el mismo nombre que nuestras tablas, conteniendo el mismo tipo y número de columnas, referenciadas de nuevo con el mismo nombre. Estas entidades definirán además los métodos básicos, lectura y escritura de cada uno de los parámetros que la entidad contiene.

El siguiente paso en la configuración de nuestro proyecto API será el de la creación de los controladores. Los controladores son los elementos que nos permitirán definir los diferentes servicios que nuestra API contendrá. Encontraremos también en el tutorial la información necesaria para entender y crear los diferentes controladores que nuestra aplicación necesite.

En nuestro caso, crearemos dos controladores diferentes, uno por cada uno de los módulos que vamos a definir: LoginController y RegisterController. Estos controladores deberán incluir las referencias a los elementos básicos de los que se sirve Symfony para poder realizar las consultas a la base de datos y crear la referencia a los repositorios.

Para la implementación de los servicios definiremos la función que realizará las acciones pertinentes. Estos módulos concretamente implementarán una única función: AttemptLogin y AttemptRegister. Estas funciones serán las encargadas de recoger la información que nuestra aplicación AngularJS les proporciona para realizar la acción pertinente. Para definir el acceso a la base de datos seguiremos un patrón común en todos los servicios: incluiremos la carga del elemento "Doctrine", que es la biblioteca de referencias que servirá de enlace entre Symfony y nuestra base de datos, y sobre ella efectuaremos la carga del modelo que corresponda. A esta carga podremos añadir los filtros que sean necesarios para acotar nuestra lista de resultados devueltos por la base de datos. Añadiremos la lógica necesaria en cada uno de los servicios sobre la respuesta que la base de datos nos proporciona, y devolveremos el valor previsto a nuestra aplicación AngularJS.

Para el enrutamiento de los diferentes servicios, es decir, las diferentes urls que hemos definido previamente en la Tabla 3, Symfony nos proporciona la directriz *@Route*, que nos permitirá definir el patrón de url esperado para que la función se ejecute. En la Figura 49 podemos observar un claro ejemplo de nuestro servicio de *login*. En ella quedan reflejados los elementos descritos anteriormente: uso de la doctrina para la entidad de usuario, aplicación de filtros en función del parámetro definido y la definición de la ruta específica para la llamada al servicio.

```
class LoginController extends Controller
{
    /**
     * @Route("/user/login", name="user_login", requirements={"user": "\s+", "password": "\s+"})
     */
    public function loginAttempt($user, $password)
    {
        $userRepository = $this->getDoctrine()->getManager()->getRepository("ApiBundle:TblUser");

        $user = $userRepository->findBy(
                    array("vchrUsername"=> $user),
                    array("vchrPassword"=> $password)
        )->first();

        if($user)
        {
            return $user->getVchrToken();
        }

        return null;
    }
}
```

Figura 49. Sprint 1: Ejemplo de implementación de un servicio en Symfony 3
(Fuente propia)

Recalcamos de esta implementación que nuestra aplicación AngularJS, con el fin de evitar el envío de información sensible sobre usuarios, codificará la contraseña introducida antes de enviarla al servicio. Como respuesta, únicamente recibirá un *token* o código de usuario, de una complejidad suficiente en ambas funciones, y será este código el que utilizaremos a lo largo de la aplicación para obtener la información relativa al usuario en los diferentes componentes. Este *token* será almacenado en las cookies del navegador, informando al usuario como dicta la política de cookies, y en el momento de la carga de la página web detectaremos si esta *cookie*

existe previamente para autenticar al usuario en el sistema sin necesidad de que éste vuelva a introducir los datos. La *cookie* será eliminada una vez el usuario haya finalizado la sesión.

9.2.3. Pruebas y validación

Durante el primer *sprint* no está prevista la realización de pruebas sobre la implementación. Recordando el proceso descrito en la

Tabla 2, durante un *sprint* realizaremos las pruebas sobre las implementaciones realizadas en el *sprint* anterior. Dado que este es nuestro primer *sprint*, dedicaremos el tiempo previsto a preparar los diferentes casos de prueba para el siguiente ciclo, e invertiremos el tiempo restante en continuar con el desarrollo.

9.3. Sprint 2

9.3.1. Análisis y Estimaciones

Procederemos con la creación de la nueva iteración en nuestro proyecto al igual que hicimos en el anterior *sprint*. En nuestro objetivo para este *sprint* partimos de la base que el usuario ya tiene los medios para acceder a nuestro sistema, bien por registro, bien por login. Por tanto buscaremos "La primera vista de nuestro *back-office* después del acceso del usuario". Incluiremos así en nuestra nueva iteración todos los elementos relevantes para el cumplimiento de nuestro objetivo. En el Apéndice III hemos incluido el listado de los elementos sobre los que trabajaremos a lo largo de este *sprint* una vez han sido analizados y estimados.

En primer lugar trabajaremos sobre el menú, elemento común para todas las vistas, y que, además de la navegación entre las diferentes vistas que compondrán nuestra aplicación, contendrá la funcionalidad del cambio de imagen de portada que los restaurantes usarán de cara a los usuarios de la aplicación.

En segundo lugar, iteraremos sobre la vista de la home. Esta vista contendrá diferentes elementos de funcionalidad independiente, por tanto, buscaremos definir cada uno de ellos como un componente, lo cual nos permitirá reutilizarlos en otras partes de la aplicación cuando sea necesario a partir de la inclusión de una única etiqueta en el código.

9.3.2. Desarrollo

9.3.2.1. Menú

Al ser un elemento que compartirán todas las vistas, encontramos dos opciones para proceder con su implementación: establecer el menú como parte de una vista principal y que cada uno de los elementos de navegación hereden de esta vista (lo cual aumentaría nuestra lógica de herencia y de posible transferencia de datos entre componentes, rompiendo así con el principio del paradigma de programación aislada por componente), o bien que el menú sea en sí mismo un componente que agregaremos a cada una de las vistas de nuestra aplicación por medio de una simple etiqueta.

Procederemos entonces con la segunda opción, y crearemos los elementos necesarios para completar el componente: vista, controlador y servicio. En este caso, la vista contendrá los

elementos de navegación, cuya lógica reside únicamente en el cambio de estado (vista). La principal funcionalidad de nuestro menú será la de ofrecer al restaurante la posibilidad de cambio de imagen representativa. Para ello, implementamos un manejo de carga de archivos dinámica mediante la directiva que AngularJS nos ofrece para ello [49].

Además, necesitaremos de la librería FileReader para la lectura y muestra de la imagen seleccionada por el restaurante de forma automática. Podemos encontrar toda la información necesaria en la documentación de FileReader [50]. Por último, crearemos un servicio que recibirá la información de la imagen cargada y la almacenará en nuestro servidor quedando así disponible hasta que sea reemplazada por una nueva.

9.3.2.2. Home

Si volvemos la vista atrás a nuestro *mock*-up de la vista de la *home* podremos identificar una serie de elementos, independientes entre ellos, que componen la vista. En la Figura 50 vemos cada uno de ellos identificado de forma independiente. Cada elemento se transformará en una implementación única, y nuestra vista de la *home* será un lienzo en blanco donde, como si de un puzle se tratase, colocaremos cada uno de los elementos creados mediante la inclusión de las etiquetas correspondientes que harán referencia a éstos.

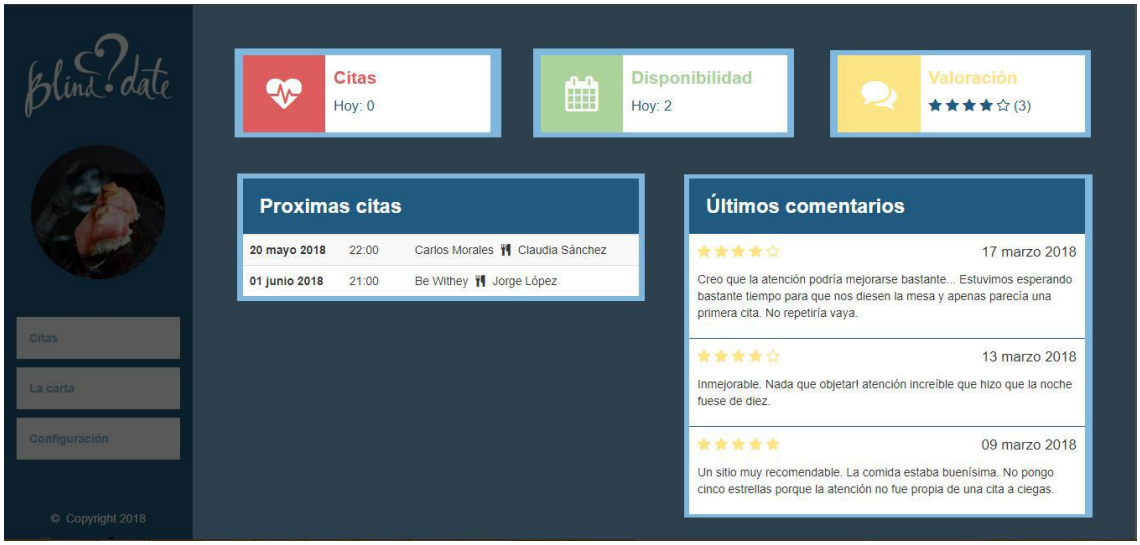

Figura 50. Sprint 2: Identificación de los componentes en la vista de inicio
(Fuente propia)

Cada uno de los componentes contendrá su propio directorio de archivos, en los cuales definiremos la vista y la lógica que contemplen. Cada uno de ellos además deberá realizar independientemente su petición de datos al servicio que corresponda para mostrarnos la información. Por último, nuestra página de menú englobará a todos ellos, aplicando las capas de estilo fijadas en nuestro *mock-up*.

La creación de los elementos como componentes, además de permitirnos la reutilización de código a lo largo de nuestra aplicación, evitará que si alguna de estas secciones se encuentra definida en varias vistas del sistema y deseamos realizar un cambio sobre ellas, debamos replicarlo a lo largo de la aplicación. La inclusión de los diferentes elementos en la vista que

deseemos se hará únicamente con la inyección de la etiqueta que hayamos definido en la creación del componente. Esta, a su vez, podrá recibir tanta información a través de la etiqueta como hayamos creído conveniente, con el fin de proveer al componente de los detalles necesarios para la carga de datos. Vemos en la Figura 51 como ejemplo la forma en la que incluimos en nuestra vista los componentes de "próximas citas" y "últimos comentarios", donde ambos de ellos recibirán de nuestra vista el usuario sobre el cual estamos interesados en cargar la información.

```
<next-dates user-id="vm.User.id"></next-dates>
<feedback-comments place-id="vm.User.id"></feedback-comments>
```

Figura 51. Sprint 2: Carga de componentes con envío por parámetro
(Fuente propia)

9.3.3. Pruebas y validación

Durante este *sprint* realizaremos las pruebas de la implementación efectuada en el ciclo anterior. Agruparemos los casos de prueba por cada una de los elementos sobre los que hemos trabajado:

9.3.3.1. Landing page

Para la validación de esta implementación, buscaremos que toda la información prevista se muestre en la página. Además, aseguraremos que la vista no se vea afectada por los diferentes tamaños de pantalla que podemos encontrar en los dispositivos. Por último realizaremos pruebas sobre la navegación de las vistas implementadas, registro y *login*. El único elemento que no cumple con este requisito es la disposición del botón de *login* en las vistas más pequeñas.

9.3.3.2. Inicio de sesión

En esta ocasión, definimos tres casos de prueba para la validación de la funcionalidad de acceso al sistema.

1. Intento de autenticación con un usuario que no existe en el sistema.
2. Intento de autenticación con un usuario existente en el sistema, pero con una contraseña incorrecta.
3. Intento de autenticación en el sistema con credenciales correctas.

El resultado de todas las pruebas fue satisfactorio. Podemos observar en la Figura 52 que, para los casos uno y dos, el sistema nos notificaba con un mensaje de error del acceso incorrecto a éste.

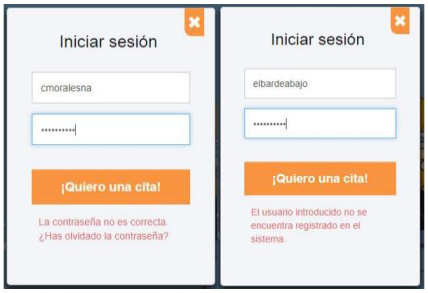

Figura 52. Sprint 2: Resultado de casos de prueba para el login (Fuente propia)

9.3.3.3. Registro

Para la validación del registro en el sistema, seremos más exhaustivos en cuanto a las pruebas realizadas:

1. La información para país/ciudad se presenta de forma correcta y contiene todas las entradas en la base de datos.
2. Los campos de registro tienen validación en cuanto a los caracteres de entrada permitidos.
3. El registro no está activo hasta que todos los campos hayan sido completados.
4. Validación de datos:
    a. El DNI no se encuentra en el sistema.
    b. El CIF no se encuentra en el sistema.
    c. El número de teléfono no se encuentra en el sistema.
    d. El nombre de usuario no se encuentra en el sistema.
    e. Las contraseñas introducidas coinciden y cumplen con la complejidad prevista.

Una vez completadas las pruebas para esta sección, obtenemos que las pruebas uno a tres obtienen un resultado satisfactorio, pero la validación de datos no se encuentra implementada. Como ejemplo, hemos podido crear varios usuarios con el mismo teléfono y nombre de usuario, lo cual además impacta en la funcionalidad de *login* y hace que nuestra aplicación no funcione. Se añadirán los elementos para la corrección de los errores en el siguiente *sprint*.

9.3.3.4. Configuración del servidor

Para completar la validación de nuestro *sprint*, comprobaremos que toda la configuración realizada en el servidor funciona correctamente. Podemos comprobar que todos los *sites* creados funcionan bajo las urls definidas, y que nuestra base de datos en el servidor contiene la estructura necesaria. Destacamos de este punto que, una vez nos encontramos navegando sobre la aplicación web, la url contiene el carácter "#" intercalado en ella. Esto es un signo propio de cualquier aplicación SPA, y es una pista relevante para todo aquél que quiera atacar nuestro sistema. Añadiremos también un nuevo elemento para el siguiente *sprint* en el que trabajaremos sobre la eliminación de este carácter de la url.

9.4. Sprint 3

9.4.1. Análisis y Estimaciones

De nuevo comenzamos el *sprint* con la creación de la iteración correspondiente. Definimos nuestro objetivo general del ciclo, esta vez tomando en cuenta los elementos que provienen del informe de errores realizado el *sprint* anterior: "Corrección de los errores del primer *sprint* y creación de dos de las vistas que hemos definido en nuestro menú, citas y carta". En el Apéndice IV podemos ver la lista reflejada de los elementos que hemos incluido en este *sprint* para completar nuestro objetivo. Veremos que los errores se encuentran vinculados a la tarea original del desarrollo, identificados con un símbolo diferente al de las nuevas tareas.

9.4.2. Desarrollo

9.4.2.1. Corrección de errores

La primera fase del *sprint* consistirá en la corrección de los diferentes errores que hemos identificado durante la fase de pruebas. Procederemos en primer lugar con la validación de los datos en el momento de registro. Para su corrección, nuestro servicio estará encargado de realizar las diferentes comprobaciones sobre la información existente para los siguientes campos: email, número de teléfono, DNI, CIF, nombre de usuario. En caso de que alguno de ellos exista previamente en el sistema en el momento del registro, nuestro servicio enviará el error como respuesta, informando al usuario del campo que no se encuentra disponible. La validación de las contraseñas se realizará en la aplicación AngularJS, ya que la información llega encriptada al servicio. Consistirá en la comprobación de que la confirmación de la contraseña coincide con la original, y que contiene caracteres especiales, letras mayúsculas, minúsculas y números intercalados.

El segundo error sobre el que trabajaremos es el de la disposición del botón de *login* en las vistas más pequeñas. Para ello, aplicaremos una carga en diferente posición del componente cuando nos encontremos con estos dispositivos, ya que con simples órdenes de estilo no será posible realizar ese cambio tan brusco.

Por último, buscaremos la manera de eliminar los caracteres especiales de la url. En una primera iteración, encontramos en la documentación de AngularJS [51] que una pequeña configuración a la hora de la definición del módulo principal, nos permitirá navegar por nuestra aplicación eliminando la almohadilla de la url.

Sin embargo, nos encontramos ahora con un nuevo problema. Aplicando esta configuración, la recarga de la página provocará que el servidor web no sea capaz de encontrar la dirección específica, mostrándonos un error 404 en pantalla. Será necesaria entonces la creación de un archivo *.htaccess* que será el encargado de escribir las reglas de redirección de forma que nuestro servidor la entienda. En otras palabras, en el momento en el que nuestro servidor acceda al directorio en busca de ese archivo inexistente, encontrará que las nuevas reglas definidas le indicarán los archivos correctos que deberá mostrar en pantalla.

9.4.2.2. *Citas*

De nuevo nos encontramos con una vista que claramente define diferentes elementos que la componen. En la Figura 53 hemos identificado cada uno de ellos. Observamos además, que uno de los componentes, "Próximas citas", será el mismo que hemos implementado anteriormente para nuestra vista de la *home*, pero con una pequeña característica que los diferencia: en esta ocasión dispondremos de un botón de acción para cada una de las citas que nos permitirá cancelarla al accionarlo. Incluiremos por tanto el mismo componente previamente definido, pero sobre el que realizaremos pequeños cambios. En primer lugar, identificaremos por el paso de un nuevo parámetro si queremos mostrar el botón de acción correspondiente. En segundo lugar, implementaremos la nueva lógica necesaria para que nuestra acción realice la llamada al servicio de cancelación de citas, el cuál recibirá el identificador de la cita en cuestión.

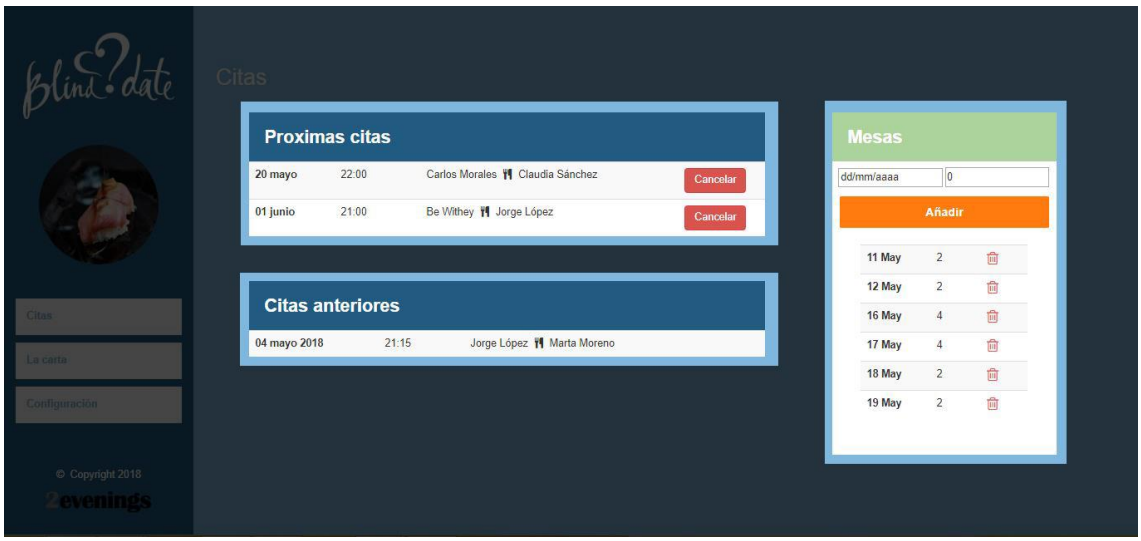

Figura 53. Sprint 3: Identificación de los componentes en la vista de citas
(Fuente propia)

El segundo componente que identificamos a partir de la vista será el de "Citas anteriores". Aunque podríamos buscar la opción de modificar de nuevo el componente existente añadiendo el envío de un nuevo parámetro que recoja las fechas de citas que buscamos, hemos preferido declararlo como un componente independiente pensando que en posibles iteraciones posteriores del proyecto, las funcionalidades definidas para próximas citas y citas anteriores puedan ser muy diferentes.

En último lugar, encontramos un nuevo componente que será utilizado para la gestión de la disponibilidad del restaurante. En él, definiremos en la parte superior la posibilidad de incluir nueva disponibilidad para una fecha dada. La disponibilidad se medirá en citas disponibles que el restaurante podrá alojar para un día determinado. Además, si el restaurante tratase de insertar nueva disponibilidad sobre una fecha que actualmente ya tiene disponible determinadas mesas, este nuevo valor se sumará al anterior.

En la parte inferior listaremos por proximidad de fecha las diferentes mesas ya publicadas, con la opción de eliminar la disponibilidad para ese día. En caso de que una cita se viese afectada, notificaríamos al restaurante de que la cita será cancelada si procede con la eliminación.

9.4.2.3. La carta

La vista de la carta permitirá al restaurante gestionar la información que ofrece a los usuarios de la aplicación móvil. Esta información, como veremos más adelante, se mostrará tanto en el listado de los diferentes restaurantes que están suscritos en la aplicación, como en el momento de elección de restaurante para la cita, lo cual empujará al restaurante a completar la información de la forma más atractiva posible.

Ya hemos hablado durante la implementación del menú de la imagen representativa del restaurante. Ahora es el momento de que el restaurante complete los detalles del tipo de restaurante que es y el menú que ofrecerá en su cena. Para ello, dividiremos de nuevo nuestra vista en los componentes correspondientes, como podemos observar en la Figura 54. En esta ocasión trabajaremos dos nuevos componentes: detalles del restaurante y menú.

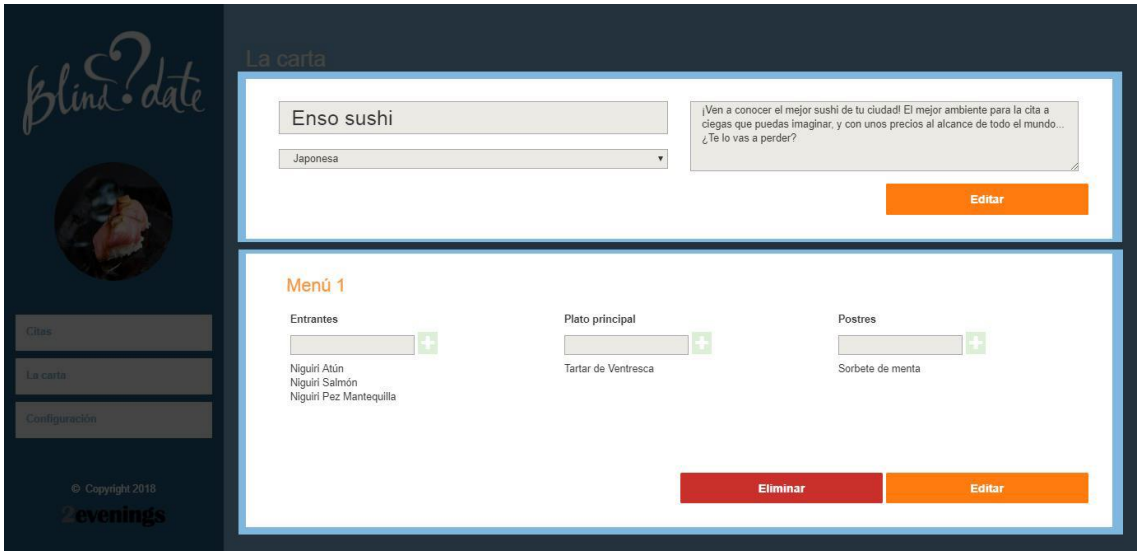

Figura 54. Sprint 3: Identificación de los componentes en la vista de la carta (Fuente propia)

El primero de los componentes manejará la información relativa al restaurante. Los tres pilares de la información a definir por parte del restaurante son: nombre, tipo de cocina y breve descripción del lugar. El tipo de cocina vendrá predefinido por un listado de tipos que el componente obtendrá a partir de la petición a un servicio. Esto nos ayudará más adelante a implementar debidamente la funcionalidad de filtrado de restaurantes por tipo en nuestra aplicación móvil.

En el segundo de los componentes es donde residirá la lógica un poco más compleja. Cada restaurante contendrá un menú asociado, que se compondrá de diferentes platos. Estos platos estarán clasificados según sean entrantes, plato principal o postres. Los platos serán almacenados de forma común, siendo posible para el restaurante elegir alguno de los platos ya existentes para incorporarlos en su menú, o definir uno nuevo. Por último, el precio del menú será fijado en 15€ para todos los restaurantes, tratando de evitar precios excesivos que hagan que los usuarios no utilicen nuestra aplicación, o que los usuarios basen su elección de restaurante en el precio del menú que ofrezcan. Empujamos así además a que los restaurantes muestren su mayor atractivo posible para obtener un mayor ratio de citas.

9.4.3. Pruebas y validación

Durante este *sprint* realizaremos las pruebas de la implementación realizada en el ciclo anterior. Agruparemos los casos de prueba por cada una de los elementos sobre los que hemos trabajado:

9.4.3.1. Menú

Para la validación del menú, obviaremos las pruebas sobre la navegación ya que no todas las vistas se encuentran implementadas. Nos centraremos por ello en la comprobación de que la carga de diferentes imágenes para un usuario se realiza de manera correcta y sin conflictos. Cerraremos y abriremos la sesión con diferentes usuarios para comprobar que la imagen mostrada es la que corresponde a cada uno de ellos. Supera las pruebas satisfactoriamente.

9.4.3.2. Home

En esta sección se han implementado diferentes componentes, por lo que será necesario realizar pruebas independientes sobre cada uno de ellos.

9.4.3.3. Home: Citas hoy

Este componente deberá mostrarnos el número de citas que están previstas para el restaurante en el día actual. Para las pruebas, añadiremos diferentes citas en la base de datos utilizando los usuarios previamente creados en las pruebas de registro y *login*. Cambiaremos las fechas para las citas, variando desde ninguna cita para el día de hoy, hasta que todas las citas ocurran hoy. Las pruebas son superadas satisfactoriamente.

9.4.3.4. Home: Disponibilidad

Este componente deberá mostrarnos la disponibilidad de mesas que nuestro restaurante tiene publicada para el día actual. Para las pruebas, añadiremos diferentes disponibilidades en el sistema, variando las fechas y la cantidad de mesas disponibles. Encontramos un posible error, pero que será necesario confirmar en la validación del siguiente *sprint*. Si existen dos entradas de disponibilidad en la base de datos para la misma fecha, provocará un error en nuestra aplicación y no podremos continuar. Asumimos que la implementación de publicar la disponibilidad no permitirá la creación de más de una entrada para un mismo día.

9.4.3.5. Home: Valoración

Este componente deberá mostrarnos el ratio de valoración que obtiene un restaurante a partir de los comentarios que le hayan realizado. Para las pruebas, insertaremos en nuestro sistema una serie de valoraciones vinculadas a citas ocurridas en el restaurante. Comprobaremos que la media de estrellas calculadas es la correcta, y que la cuenta del número de valoraciones situada a la derecha también lo es. Supera las pruebas satisfactoriamente.

9.4.3.6. Home: Próximas citas

Este componente deberá listar las citas relativas al restaurante que ocurrirán en los días posteriores al día actual. Para ello, teniendo ya una serie de citas creadas para la validación del anterior componente, modificaremos la fecha prevista entre días anteriores, actual y días próximos para asegurar que muestra la información prevista. Supera las pruebas satisfactoriamente.

9.4.3.7. Home: Últimos comentarios

Este componente deberá listar los últimos comentarios realizados hacia el restaurante en anteriores citas. Para la realización de las pruebas, crearemos diferentes entradas en nuestra base de datos con diferentes comentarios y su correspondiente valoración hacia el restaurante. Encontramos un error en lo relativo a la relación entre la valoración numérica y la descriptiva. En algunos casos, la valoración numérica (reflejada en estrellas) no corresponde a la entrada de comentario descriptivo. Crearemos este nuevo elemento de trabajo para ser incluido en el siguiente *sprint*.

9.5. Sprint 4

9.5.1. Análisis y Estimaciones

Comenzamos ahora con el último *sprint* que involucra desarrollo para nuestra aplicación de gestión. En el Apéndice V encontramos nuestra nueva iteración creada, con los últimos elementos de trabajo que definimos anteriormente, ya analizados y estimados. Además, incluimos el error notificado en el *sprint* anterior durante la fase de validación. En este *sprint* nuestro objetivo es bastante claro: "Concluir la aplicación web". Esto se traduce en que nos centraremos en trabajar en nuestra última vista de la aplicación: configuración.

Esta vista buscará definir los detalles de la cuenta de los restaurantes. Si echamos la vista atrás, durante el primer *sprint* desarrollábamos una vista de registro donde el restaurante debía rellenar una serie de información relativa a éste. Pues bien, ahora es el momento de poder gestionar toda esa información.

9.5.2. Desarrollo

9.5.2.1. Corrección de errores

El error que ha sido añadido para su corrección en este *sprint* hace referencia a la relación que existe entre la valoración como comentario y numérica. La funcionalidad de la valoración por parte del usuario todavía no ha sido implementada, pero se ha previsto el patrón de almacenamiento de información que ésta tendrá. Esto nos daba la capacidad de implementar la API que recogía los datos. El problema surgía derivado de la consulta a la base de datos, ya que en una primera instancia únicamente cargábamos los datos existentes para el restaurante, de tipo valoración y de tipo comentario, agrupados por la referencia a la cita. El caso en el que la información se cruzaba era cuando ambos usuarios de la cita realizaban el comentario y la valoración, lo que provocaba que la consulta nos devolviese las cuatro entradas, pero no teníamos forma de identificar cómo correspondían entre ellos.

Añadiendo la agrupación de resultados también por la referencia de usuarios soluciona completamente el problema y, además, los resultados de la consulta se hacen incluso más manejables desde nuestro servicio ahora.

9.5.2.2. Configuración

Son cuatro las funcionalidades básicas que extraemos de nuestro *mock-up* sobre la vista: editar datos personales, modificar la contraseña, gestionar la cuenta Premium y cancelar la cuenta. Durante la fase de análisis hemos hecho especial hincapié en el apartado de cuenta Premium. Esta funcionalidad está prevista como una de las fuentes de ingresos, donde los restaurantes

aumentarán el coste de su suscripción con el fin de obtener una serie de funcionalidades no disponibles para la suscripción general.

Encontramos que en esta primera versión del proyecto, todas las funcionalidades que hemos ido implementando a lo largo de los diferentes *sprints* se hacen necesarias como un mínimo de atractivo y de funcionalidad de la aplicación. Por tanto, optamos por incluir este elemento en nuestro menú, pero mostraremos un mensaje cuando el restaurante acceda a esta vista informándole de que esta funcionalidad todavía no se encuentra disponible en nuestro sistema. Podemos ver el resultado de esta implementación en la Figura 55.

Para su implementación seguiremos nuestro patrón de creación de componentes. Por el momento, únicamente contendrá el archivo de definición del componente en sí, sumado a la pequeña vista que declaramos con el mensaje informativo que deseamos hacer llegar a nuestro usuario.

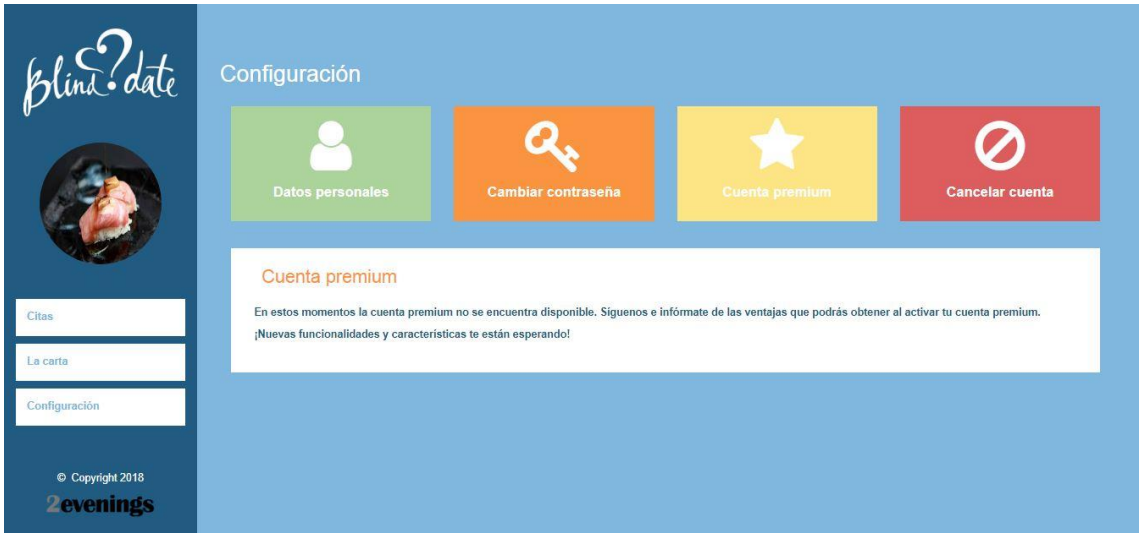

Figura 55. Sprint 4: Funcionalidad Premium no disponible
(Fuente propia)

Para proceder con la implementación de la vista, en primer lugar será necesario definir el menú secundario que encontramos en la parte superior. Estos elementos nos permitirán navegar por las diferentes configuraciones sobre las que podremos trabajar. En esta ocasión el menú no trabajará sobre cambio de estados, ya que esto implicaría la definición de diversas vistas para un mero cambio de carga de componente. Por tanto, este menú estará definido directamente en nuestro archivo de vista, y cada uno de los diferentes elementos inferiores serán definidos en componentes externos.

Así, en nuestra vista principal trabajaremos con cambios de estado internos sobre una enumeración que definiremos previamente. La idea general es establecer una serie de valores posibles en cuanto al estado de la vista (ej. Configuración General, Configuración Contraseña…) y será la acción sobre los botones la que modificará el valor de este estado y actualizará el componente que mostramos por pantalla.

Toda esta explicación se hará mucho más comprensible si nos fijamos en la Figura 56. En ella podremos observar perfectamente el punto anterior explicado acerca de los estados de la vista. En primer lugar, nos apoyaremos en una de las directivas que AngularJS nos brinda: "ng-if". Esta directiva nos permitirá definir una pequeña lógica en la vista para determinar si ese bloque debe mostrarse en pantalla o no. En nuestro caso, únicamente comprobamos para cada uno de los bloques si nuestra variable de estado corresponde con el tipo designado para ese componente. Podemos encontrar toda la información sobre esta directiva en la documentación de AngularJS [52].

```
<div ng-if="vm.ConfigurationView.Type == vm.ConfigurationViewModes.Personal">
    <configuration-personal-data user="vm.User"></configuration-personal-data>
</div>

<div ng-if="vm.ConfigurationView.Type == vm.ConfigurationViewModes.Password">
    <configuration-change-password user-id="vm.User.id"></configuration-change-password>
</div>

<div ng-if="vm.ConfigurationView.Type == vm.ConfigurationViewModes.Premium">
    <configuration-premium></configuration-premium>
</div>

<div ng-if="vm.ConfigurationView.Type == vm.ConfigurationViewModes.Cancel">
    <configuration-cancel-account user-id="vm.User.id"></configuration-cancel-account>
</div>
```

Figura 56. Sprint 4: Lógica de componentes en la vista de configuración
(Fuente propia)

Por último, necesitaremos definir cada uno de los componentes mencionados. Habremos observado en la Figura 56 que algunos de ellos tendrán prevista la recepción de la referencia de usuario por parámetro. Esto se hace necesario ya que cada uno de los componentes deberá realizar su acción prevista sobre nuestro usuario en concreto. Así, por ejemplo, el componente de gestión de datos personales definirá la vista, pero necesitará de un servicio en nuestra API el cual, mediante el envío de la referencia del usuario, sea capaz de enviarnos los datos del usuario solicitados como respuesta. También será necesario el envío de la referencia a esta API cuando se produzca una modificación en los datos y deseemos guardar los cambios.

La funcionalidad de cancelar cuenta supondrá la cancelación inmediata de todas las citas activas o pendientes que se encuentren asociadas a este restaurante. La disponibilidad y el menú del restaurante quedarán también suprimidos de nuestro sistema. La información relativa al registro se mantendrá ya que será necesario para la relación que existe entre la entidad de la cita y los restaurantes en base de datos, permitiendo así mantener el historial de citas en la vista de la aplicación de usuario.

9.5.3. Pruebas y validación

9.5.3.1. Corrección de errores

En primer lugar, durante nuestra fase de validación reiteraremos sobre las pruebas realizadas en el *sprint* anterior donde pudimos identificar una serie de errores.

Registro

En nuestro ciclo anterior pudimos detectar que los campos de registro no cumplían una validación contra los datos existentes previamente en la base de datos. Esto se traducía en que podíamos crear usuarios con información duplicada, provocando un error en nuestro sistema. Realizando de nuevo las pruebas sobre esta funcionalidad, observamos que se han añadido mensajes informativos en la vista cuando alguno de nuestros principales campos está duplicado: email, CIF, DNI o nombre de usuario. En la Figura 57 podemos observar un ejemplo de ello, sobre la validación que se ha realizado para la comparación de las contraseñas introducidas.

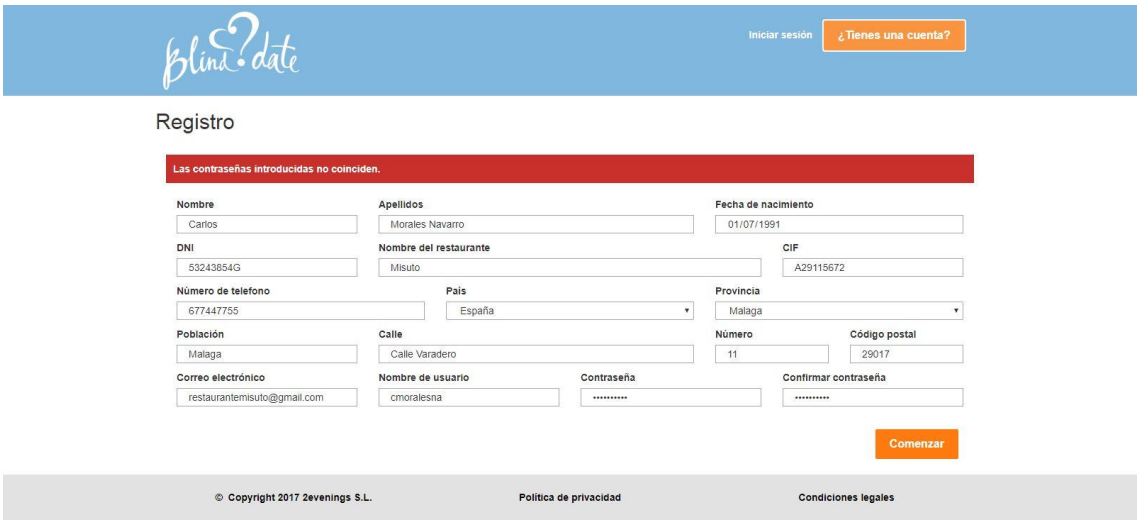

Figura 57. Sprint 4: Validación de datos erróneos en el registro
(Fuente propia)

Landing page

Para este elemento nos encontrábamos con que en las vistas más pequeñas, el botón de *login* se descuadraba completamente en nuestra pantalla. Cargando de nuevo la página en una vista *responsive*, podemos observar que el botón de *login* se va acomodando correctamente en las diferentes dimensiones de la pantalla.

Caracteres en la URL

Confirmamos que la url ahora no contiene el carácter de "#", y que la recarga de la página se realiza correctamente a lo largo de la aplicación.

9.5.3.2. Citas

En esta sección se han implementado diferentes componentes, por lo que será necesario realizar pruebas independientes sobre cada uno de ellos.

9.5.3.3. Citas: Próximas citas

Para este componente ya se realizó previamente la validación de carga correcta de citas. Por tanto, será necesaria la comprobación del correcto funcionamiento de la nueva funcionalidad implementada: cancelar citas. Para ello, nos centraremos en la cancelación de alguna de las citas existentes actualmente y comprobamos que el estado de la cita es modificado satisfactoriamente en la base de datos.

9.5.3.4. Citas: Citas anteriores

En esta ocasión será necesaria la modificación de las citas existentes en la base de datos, variando la fecha en la que cada una de ellas ocurre, para asegurar que el componente incluya únicamente las anteriores al día actual. Además, haremos pruebas sobre el cambio de estados de las citas, para asegurar que las citas incluidas no comprenden citas canceladas, pendientes o activas, y únicamente son listadas las que se realizaron satisfactoriamente. Supera la prueba con éxito.

9.5.3.5. Citas: Disponibilidad

Serán dos las secciones donde tendremos que realizar las pruebas para esta funcionalidad. En primer lugar, realizaremos las pruebas necesarias sobre la creación de nuevas disponibilidades:

1. Publicación de disponibilidad sobre una fecha libre.
2. Publicación de disponibilidad sobre una fecha que contiene previamente información. En este caso nuestro sistema debería de mantener una única entrada para la fecha en cuestión, y la disponibilidad total será la suma de ambas.

Para la comprobación de que las pruebas son realizadas de forma satisfactoria, haremos uso del listado de disponibilidad publicado, validando también éste a su vez.

1. La disponibilidad listada contiene únicamente fechas posteriores a la fecha actual. En este listado no esperamos ver la disponibilidad del restaurante del mes anterior.
2. Una vez publiquemos nuevas disponibilidades, esperamos que nuestro listado se actualice de forma automática añadiendo el nuevo elemento en ella (o la suma de mesas disponibles).
3. Eliminar una publicación disponible. Esperamos también aquí que el listado sea actualizado de forma automática.

En este caso, será necesaria la creación de un *bug* relacionado con la recarga automática en el listado de disponibilidad. Toda la funcionalidad de gestión de la disponibilidad supera la validación con éxito, pero una vez añadimos o eliminamos alguna disponibilidad en nuestro listado, no se realiza de forma automática la actualización. Es necesaria la recarga de la página para ver la información correctamente.

9.5.3.6. Carta: Detalles del restaurante

Para este componente será necesario comprobar, en primer lugar, que el listado de tipos de restaurante contiene todos (y únicamente) los que se encuentran disponible en nuestra base de datos. Además, comprobaremos que la información se carga y se muestra de forma correcta, y la modificación de ésta se guarda correctamente en la base de datos. Supera todas las pruebas satisfactoriamente.

9.5.3.7. Carta: Menú

Para la validación del menú será necesario realizar previamente la validación de los platos disponibles. Como mencionamos durante el proceso de desarrollo, los platos son almacenados independientemente en nuestra base de datos y relacionados con el menú que corresponda.

Además, cada plato formará parte de un tipo diferente. Por tanto, en primer lugar deberemos comprobar que los listados de platos contienen las entradas de la base de datos, y cada uno de ellos se encuentra en el lugar que le corresponde. En segundo lugar, si introducimos un nuevo plato, no existente en la base de datos, este nuevo plato será creado y almacenado en la lista en el momento de guardado. Por último, deberemos validar la funcionalidad del menú de añadir y eliminar diferentes platos de éste, guardando las modificaciones correctamente en la base de datos. Supera todas las pruebas satisfactoriamente.

9.6. Sprint 5

9.6.1. Análisis y Estimaciones

Habiendo concluido el desarrollo en lo relativo a nuestra aplicación de gestión, es el momento de dar comienzo a la implementación de la aplicación móvil. Para comenzar el *sprint*, como hemos hecho durante los anteriores ciclos, crearemos la nueva iteración en TFS, en la cual incluiremos todos los elementos sobre los que trabajaremos y estimaremos después del análisis de éstos.

El desarrollo de nuestra aplicación móvil ha sido dividida en dos grandes bloques, buscando completar todo lo relativo a ésta en los siguientes dos *sprints*, dando entonces por concluido el desarrollo de nuestro proyecto. Para cubrir el primero de los dos bloques, nuestro objetivo para el *sprint* actual será: "Creación del proyecto Ionic e implementación de las vistas de gestión de la cuenta". En estas vistas incluiremos desde la vista de login, pasando por la del registro y completamos el *sprint* con la implementación de las opciones de configuración de la cuenta.

Además, debemos recordar que durante la fase de validación del *sprint* anterior se ha detectado un error en la implementación realizada. Por tanto, incluiremos la corrección de este error en el *sprint* actual. Encontraremos el listado completo de las tareas para nuestro *sprint* y sus estimaciones en el Apéndice VI.

9.6.2. Desarrollo

9.6.2.1. Corrección de errores

Comenzaremos el *sprint* con la corrección del error identificado durante la fase de validación. Este error se encuentra vinculado a nuestro módulo de gestión de disponibilidad. La gestión de disponibilidad esta implementada como la integración de dos elementos independientes: por una parte el formulario para la creación de nueva disponibilidad, y por otra el listado de la disponibilidad actual. Además, aparece un tercer agente que es la funcionalidad de eliminación de un elemento de nuestra lista.

Cada una de las disponibilidades funciona correctamente de forma independiente. El error reside en que no existe una comunicación entre las funcionalidades de añadir y eliminar elementos hacia el listado. Esto implica que nuestro listado no tiene conocimiento de que la lista ha cambiado, y que, por tanto, debe volver a cargar el contenido.

Para la gestión de la transferencia de esta información, angular nos proporciona la posibilidad del lanzado de eventos. Básicamente, la idea es que al eliminar o añadir un elemento de

nuestra lista, notificaremos a nuestro módulo principal que ese evento ha sido realizado. Por otra parte, nuestra lista tendrá implementada la escucha de los dos posibles eventos. Es decir, que cuando el módulo principal reciba la información de que ese evento se ha realizado y se lo comunique al resto de módulos, nuestra lista reconocerá ese evento y actualizará la información. Encontramos un artículo muy interesante que nos ofrece una explicación y demostración de las tres formas diferentes que AngularJS nos ofrece para la propagación de eventos [53].

En la Figura 58 podemos observar el código añadido en nuestra solución para la gestión del evento de publicar una nueva disponibilidad. En la parte superior podemos observar como al añadir la disponibilidad, le indicamos a nuestro sistema que ha sido añadida mediante la propagación del evento "availability-added". En la parte inferior nuestra lista estará a la escucha de ese evento, y, en el momento en el que sea notificada de que ha ocurrido, realizará la nueva carga de los elementos para publicarlos en la lista.

Figura 58. Sprint 5: Propagación y manejo de eventos en AngularJS
(Fuente propia)

9.6.2.2. Configuración de Ionic

Ionic es el *framework* que utilizaremos para el desarrollo de nuestra aplicación. Como ya vimos durante el apartado de diseño, nos permitirá la creación de nuestra aplicación para servirla en multiplataforma sin necesidad de realizar el desarrollo en los lenguajes nativos de cada uno de los distribuidores.

Nuestro primer paso consistirá entonces en la configuración de este *framework* para poder comenzar a trabajar con él. La documentación de Ionic nos explica de forma detallada los pasos que debemos seguir para la correcta configuración y la creación de nuestro primer proyecto [54]. En el artículo se detalla, además, la inclusión de las dependencias que será necesario incluir en nuestro equipo para que el *framework* pueda funcionar correctamente.

Procedemos así con la creación de nuestro proyecto. Podemos observar en la documentación como mediante la consola de nuestro sistema operativo podremos dar la instrucción a Ionic para crear el sistema de ficheros de nuestro nuevo proyecto. Además, Ionic define por defecto una serie de plantillas de proyectos, las cuales contendrán determinadas funcionalidades y pantallas creadas. Si nos fijamos en el listado de las posibilidades ofrecidas, observamos como la aplicación generada a partir de la plantilla "super" nos proporcionará un resultado que se aproxima mucho al resultado que estamos buscando.

A partir de este punto, seremos capaces de ejecutar de forma local nuestra aplicación. Será ejecutada en nuestro explorador, el cual actuará como dispositivo. Debemos configurar nuestro explorador para que muestre la pantalla como si de un móvil se tratase (nos dará a elegir entre diferentes dispositivos) para que la resolución sea la adecuada y el diseño de la aplicación se muestre como el resultado final.

Nuestros pasos por la aplicación consistirán en identificar cuáles son las vistas que incluye de primera mano la aplicación y que debemos modificar para que se adapten a nuestros diseños, y completar la aplicación con las vistas necesarias.

Página de inicio

La primera versión de nuestra aplicación creada a partir de la plantilla contiene una serie de vistas informativas en el momento de apertura de ésta. En nuestro caso, querremos que la pantalla de inicio sea única. En nuestra fase de diseño no hemos previsto ningún *mock-up* para esta vista ya que desconocíamos que fuese necesario. En la Figura 59 encontraremos el resultado final que hemos implementado como pantalla de inicio. A la izquierda, podremos observar la vista que obteníamos al cargar la aplicación sin modificación alguna de la estructura inicial.

Para su implementación habrá sido necesaria la eliminación de las sucesivas vistas que nuestra aplicación cargaba a modo de *slides*, y la adaptación del diseño de la pantalla restante al boceto que hemos generado durante la implementación.

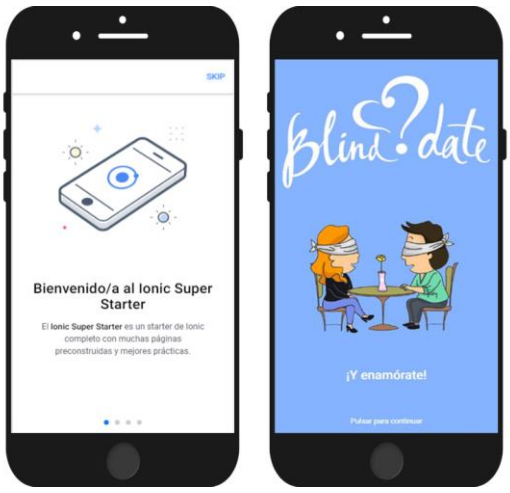

Figura 59. Sprint 5: Resultado de la vista de inicio
(Fuente propia)

Login

Para la vista del *login* únicamente será necesario trabajar sobre los colores que la aplicación define por defecto. En nuestro caso, trabajaremos con la misma gama de colores que definíamos en la aplicación de gestión, remarcando así la integración entre ambas aplicaciones. Podremos observar los estilos aplicados en la Figura 60.

Además de las modificaciones sobre la parte de diseño, será necesario trabajar sobre el código ejecutado sobre esta vista. Ionic, como ya mencionamos en el apartado de diseño, trabaja con

Angular como base de su código. La versión de ésta es posterior a la que hemos aplicado en nuestro *back-office*, por lo que no podremos aplicar el mismo código exactamente. Pero realizaremos las modificaciones necesarias para que nuestra nueva aplicación realice la llamada a la API previamente definida para el restaurante, y manejar así el sistema de acceso a la aplicación móvil.

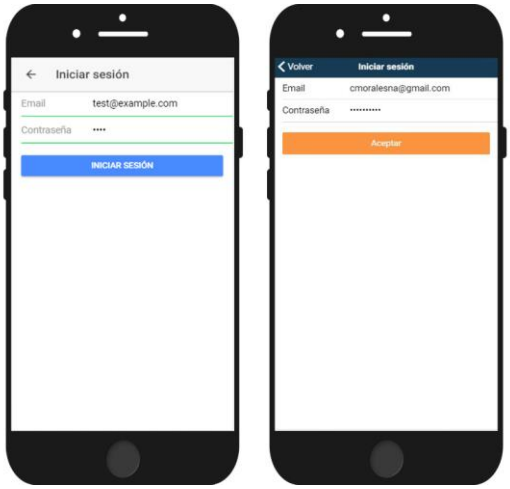

Figura 60. Sprint 5: Resultado de la vista de inicio de sesión
(Fuente propia)

Registro

Con la vista de registro nos encontramos con el mismo caso que en el apartado anterior. El proyecto actualmente contiene definida la vista, por lo que únicamente debemos iterar sobre los diferentes datos que nuestro registro necesita, y aplicar las modificaciones tanto en el diseño como el código que harán que la vista esté completa.

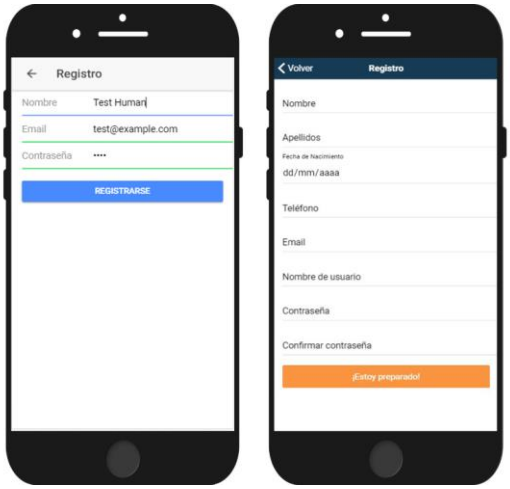

Figura 61. Sprint 5: Resultado de la vista de registro
(Fuente propia)

Configuración

La vista de ajustes también se encontrará previamente definida en la plantilla. Para la adaptación de esta pantalla, en primer lugar, deberemos iterar sobre el diseño general de nuestra aplicación. Como ya hemos mencionado, será necesario aplicar una serie de

modificaciones en los colores que la aplicación define, pero además, deberemos aplicar los cambios respectivos al menú de navegación que se encuentra en la parte inferior de la pantalla. Para ello, definiremos un nuevo elemento no existente en la plantilla inicial, y modificaremos los iconos de cada una de nuestras ventanas para adaptarse al resultado esperado. A continuación, iteraremos sobre los diferentes elementos que nuestra configuración contendrá listados.

Configuración: Editar perfil

De nuevo nos encontramos con un sistema de acceso y una pantalla previamente existente en la plantilla, por lo que será necesario trabajar sobre el diseño de la vista y la funcionalidad de ésta. Basaremos el diseño en el que utilizamos previamente para la vista del registro, y la funcionalidad será prácticamente la misma, cambiando el servicio de consulta por el de modificar los datos en lugar de almacenar datos nuevos.

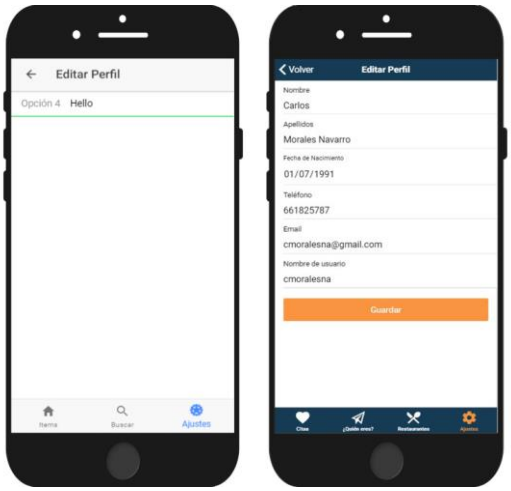

Figura 62. Sprint 5: Resultado de la vista de editar perfil
(Fuente propia)

Configuración: Cambiar contraseña

Esta vista no se encontrará definida previamente en el sistema, pero duplicaremos la lógica existente en la aplicación utilizada para la vista de editar perfil. Obtendremos, así, la navegación a una vista sobre la cuál necesitaremos iterar en los campos solicitados al usuario y en el servicio al que realizamos la petición, utilizando de nuevo el servicio previamente definido para los restaurantes en el *back-office*. Podemos ver el resultado de nuestra vista fiel al *mock-up* que definíamos en el apartado de diseño, Figura 28.

Configuración: Usuarios bloqueados

Esta vista no se encontrará definida previamente en el sistema, pero duplicaremos la lógica existente en la aplicación utilizada para la vista de editar perfil. En esta ocasión será necesaria la creación de nuevos elementos dentro de la vista, que se compondrá de dos secciones.

La primera de ellas, será un listado con los usuarios que actualmente se encuentran bloqueados, dándole al usuario la habilidad de eliminar ese bloqueo en cualquier momento. Este bloqueo implicará que si existe algún usuario en nuestro sistema con el número de teléfono provisto como bloqueo, no se tendrá en cuenta como posible candidato a una cita. Necesitaremos de dos nuevos servicios para esta sección: un primer servicio que nos provea

de la información bloqueada actualmente, y un segundo servicio que nos permita eliminar alguna de las entradas existentes.

La segunda sección, proporcionará al usuario un campo donde introducir un número de teléfono a bloquear, y un botón que ejecutará la acción. Se hará necesario definir un tercer servicio para esta funcionalidad que incluirá este nuevo dato en los elementos bloqueados. Deberemos asegurar que la entrada no existe previamente para evitar posibles duplicados.

Configuración: Cancelar cuenta

Para esta funcionalidad optamos por no definir una vista, ya que únicamente debemos informar al usuario de las consecuencias de la cancelación de la cuenta, y de que procederá con la acción. Así, decidimos que la implementación de una alerta (*modal* en términos más actualizados y técnicos) donde mostraremos al usuario todo lo mencionado anteriormente, siendo esto más que suficiente.

El servicio que realizará la acción de cancelar la cuenta ya se encontrará previamente definido en el sistema. Únicamente será necesario realizar una serie de modificaciones para que proceda de una manera u otra, según si el usuario a cancelar sea un restaurante o un candidato, ya que las entidades implicadas no son compartidas al cien por cien.

Configuración: Cerrar sesión

De nuevo optaremos por la implementación de un *modal* informativo al usuario donde pediremos la confirmación del cierre de sesión. En esta ocasión la información relativa a la sesión no será almacenada en *cookies* (ya que una aplicación no dispone de ellas). Será necesario el almacenamiento del *token* provisto al inicio de sesión en la memoria del teléfono, y, por tanto, al cerrar la sesión será de ahí de donde eliminemos esta información. Podemos encontrar toda la información relativa al almacenamiento de datos en la documentación de Ionic [55].

9.6.3. Pruebas y validación

9.6.3.1. Corrección de errores

En primer lugar, durante nuestra fase de validación reiteraremos sobre las pruebas realizadas en el *sprint* anterior donde pudimos identificar una serie de errores.

Citas: Disponibilidad

En esta ocasión identificábamos que las funcionalidades de cada uno de los elementos estaban definidas correctamente, pero la actualización de los datos en pantalla no se realizaba de forma automática. Así, las únicas pruebas a realizar serán las de la eliminación e inclusión de diferentes disponibilidades. Observamos que ahora la lista se actualiza de forma automática con los elementos afectados.

9.6.3.2. Configuración: Menú

Para esta sección tendremos que asegurar que la navegación dentro del sub-menú creado para la vista de configuración funciona de manera correcta. Supera con éxito las pruebas realizadas.

9.6.3.3. Configuración: Datos personales

Los casos de prueba que utilizaremos para la validación de esta sección serán los mismos que propusimos para el registro de un nuevo candidato, exceptuando los casos relativos a la contraseña, ya que la funcionalidad vuelve a afectar directamente a los campos principales de nuestra aplicación. De nuevo, comprobamos que las diferentes validaciones de campos se encuentran implementadas, con sus correspondientes mensajes de alerta hacia el usuario sobre el campo afectado.

9.6.3.4. Configuración: Cambiar contraseña

Para la validación de esta funcionalidad serán dos nuestros principales objetivos:

1. Asegurar que la comprobación de la contraseña actual se efectúa de manera correcta.
2. Asegurar que la nueva contraseña introducida cumple los requisitos de caracteres, y que la confirmación de la nueva contraseña coincida.

Supera las pruebas realizadas con éxito.

9.6.3.5. Configuración: Cuenta Premium

En este caso no habrá validación necesaria ya que la funcionalidad no se ha llegado a implementar. Comprobaremos que el mensaje mostrado para el usuario es el esperado.

9.6.3.6. Configuración: Cancelar cuenta

Para la validación de esta funcionalidad, será necesario disponer de ciertos elementos previos definidos: disponibilidad creada, citas activas en el restaurante en cuestión, menú disponible…

1. Aseguraremos que al cancelar la cuenta el estado del usuario cambia en nuestro sistema y no puede acceder a él.
2. Aseguraremos que la disponibilidad del restaurante queda eliminada de nuestro sistema.
3. Aseguraremos que las citas asociadas al restaurante pasan a estado de cancelación.
4. Aseguraremos que el menú generado por el restaurante queda eliminado de nuestro sistema.

Supera las pruebas realizadas con éxito.

9.7. Sprint 6

9.7.1. Análisis y Estimaciones

Ha llegado el momento de comenzar con nuestro último *sprint* de desarrollo. Durante este *sprint* nuestro objetivo será: “Concluir la aplicación móvil”. Esto será un gran reto ya que son varias las vistas y nuevas funcionalidades que todavía tenemos que implementar, pero con una buena planificación podremos alcanzar nuestro objetivo.

Creamos entonces nuestra última nueva iteración, añadiéndola a nuestro proyecto y moviendo los últimos elementos que encontramos en nuestro listado. Recordamos que del anterior

sprint no hemos recibido ningún *bug*, lo que significa que podremos centrarnos única y exclusivamente en el nuevo desarrollo. Podemos observar nuestra nueva iteración con los elementos que compondrán el *sprint*, sus tareas y estimaciones en el Apéndice VII.

Los cuatro pilares sobre los que trabajaremos en el *sprint* serán: listado de citas del usuario, listado de restaurantes del sistema, compleción del formulario y creación de una nueva cita.

9.7.2. Desarrollo

9.7.2.1. Listado de citas

En primer lugar identificamos que en las vistas de plantilla se incluye una que nos ofrece un listado de elementos. Por tanto, trabajaremos sobre la modificación de esta vista para que se ajuste a nuestro diseño esperado. En nuestro *mock-up* definíamos el listado dividido en dos secciones: citas activas y citas anteriores. Simularemos así, las funcionalidades de "próximas citas" y "citas anteriores" que definíamos anteriormente en nuestro *back-office*., Esto, además, implica que tendremos la opción de reutilizar los servicios de la API que habíamos definido anteriormente, aunque será necesario realizar un pequeño ajuste en la lógica. Debemos ahora identificar qué tipo de usuario realiza la petición, y, en función de ello, buscaremos las citas donde el usuario enviado sea anfitrión o candidato a la cita. En la Figura 63 podemos observar la pantalla definida en la plantilla a la izquierda, y, a la derecha, el resultado después de nuestra implementación.

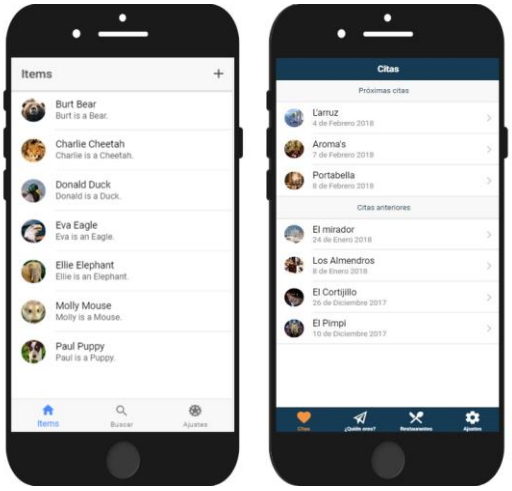

Figura 63. Sprint 6: Resultado de la vista de listado de citas
(Fuente propia)

Una vez tenemos nuestro listado de citas implementado, será necesaria la implementación de la vista del detalle de la cita. De nuevo, nos encontramos con que la vista existe en nuestra plantilla, por lo que será necesario, en primer lugar, la modificación del diseño para adaptarlo al resultado esperado, y, en segundo lugar, iteraremos sobre el código para aplicar las funcionalidades relativas al detalle de la cita que sean necesarias.

Esta vista ahora recibirá los datos de una cita, y será necesario discernir sobre las opciones que mostraremos al usuario. En primer lugar, si la cita ya ha sucedido el usuario debe tener la opción de valorarla si todavía no lo ha hecho. En segundo lugar, si la cita no ha sucedido pero ha sido aceptada, el usuario tendrá la opción de cancelar la cita. Por último lugar, si la cita no

ha sido aceptada todavía, el usuario tendrá la opción de modificarla y/o aceptarla. Estas funcionalidades irán dadas por los diferentes botones de acción que mostraremos en pantalla según el caso.

En la Figura 64 observamos el ejemplo de la vista de detalle de una cita que ya ha sido aceptada. En la parte izquierda de nuevo la implementación ofrecida a partir de la plantilla, y, a la derecha, el resultado de nuestra implementación.

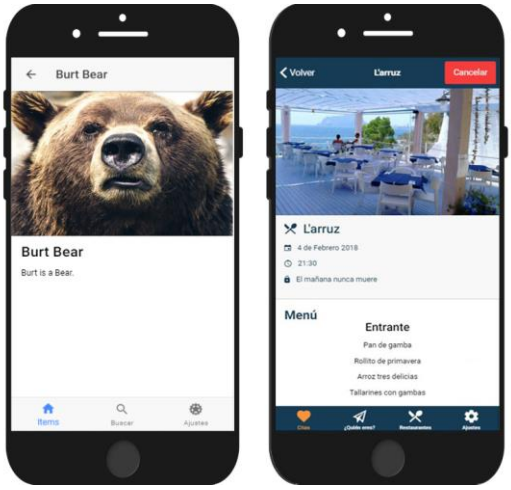

Figura 64. Sprint 6: Resultado de la vista del detalle de la cita
(Fuente propia)

Sobre los diferentes servicios necesarios para esta vista, en primer lugar, recalcaremos que la función de cancelación de cita fue implementada previamente para la aplicación de gestión, y por tanto, será posible realizar la llamada al mismo servicio desde la aplicación móvil. Además de éste, será necesaria la existencia de otros dos servicios: aceptación y modificación de cita. Para la aceptación de la cita trabajaremos únicamente con la referencia de la cita, cambiando el estado de ésta a activo en nuestro sistema. Para la modificación de la cita, será necesario además enviar al servicio de nuevo los datos de la cita para que sean sobrescritos con los proporcionados en esta ocasión por el usuario.

9.7.2.2. Listado de restaurantes

La plantilla nos ofrece una vista creada que busca iterar sobre un listado añadiendo una barra de búsqueda superior. Esta vista será idónea como punto de partida para nuestro listado de restaurantes. De nuevo, nuestro primer paso será realizar la modificación pertinente de diseño para obtener el resultado establecido en nuestros *mock-ups*.

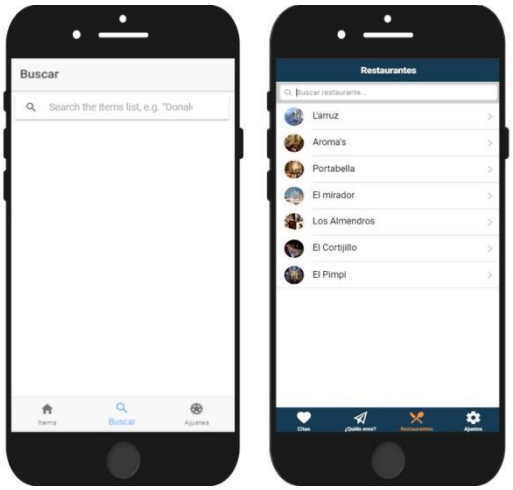

Figura 65. Sprint 6: Resultado de la vista de listado de restaurantes
(Fuente propia)

La muestra de resultados se basa en el valor introducido en la barra de búsqueda. En un principio este filtro se aplica únicamente a uno de los campos (en concreto el nombre) de todos los elementos que componen la lista. Deberemos iterar sobre este filtro para que nos muestre la lista completa en caso de que no hayamos introducido ningún valor de búsqueda, y para que el valor de búsqueda se aplique a los campos de: nombre, ciudad y tipo de restaurante.

Será necesaria, además, la implementación de un nuevo servicio en nuestra API que nos proporcione toda la información de los restaurantes definidos en el sistema. La Figura 65 nos muestra la vista que obteníamos a partir de la creación de la plantilla, y, a la derecha, el resultado de la vista después de haber realizado nuestra implementación.

Por último, para completar esta sección deberemos realizar una serie de modificaciones en la implementación de la vista de detalle anteriormente realizada. Al seleccionar en nuestra lista de restaurantes cualquiera de ellos, navegaremos a la vista de detalle (la que anteriormente utilizamos para mostrar el detalle de la cita).

Dado que la información a mostrar será casi exactamente la misma que en la ocasión anterior, optamos por reutilizar todos los elementos, y discernir si la información enviada pertenece a una cita o a un restaurante, para mostrar los detalles de la cita o la descripción del restaurante en su lugar, manteniendo el resto de la vista para ambos casos intacta. En la Figura 66 podremos observar en la implementación de la misma vista para ambos casos, a la izquierda el caso del restaurante y a la derecha el caso de una cita.

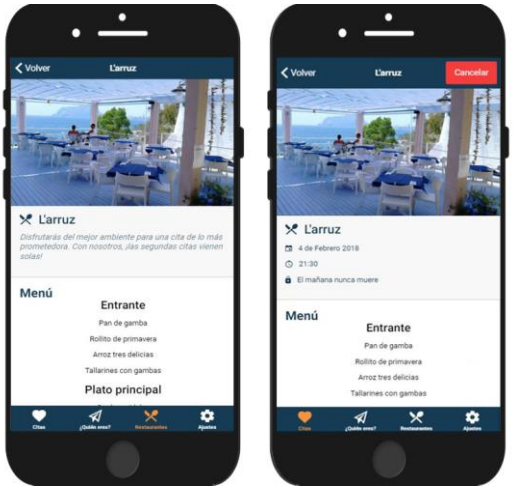

Figura 66. Sprint 6: Implementaciones de la vista de detalle
(Fuente propia)

9.7.2.3. Formulario

Para la implementación del formulario será necesario realizar un pequeño estudio previo sobre las diferentes preguntas que las plataformas similares a la nuestra realizan a sus usuarios. En nuestro caso buscaremos centrarnos en las preguntas de carácter personal y social, obviando las preguntas de índole física. No buscamos definir un listado definitivo de preguntas, ya que esto supondría un auténtico estudio sociológico sobre los patrones que definen la compatibilidad entre dos personas, y esta tarea en sí misma es objeto de un TFG independiente. La obtención de un listado definitivo a partir de un estudio será sin duda uno de nuestros elementos a añadir en el trabajo futuro. Una vez hemos definido la lista de preguntas que utilizaremos en nuestra primera versión funcional, incluiremos toda esta información en nuestra base de datos.

El primer paso en la implementación del formulario será el de la creación de la vista. No encontramos ninguna vista de las que incluye la plantilla que se ajuste a nuestro requerimiento, por tanto será necesario crear una vista desde cero. Basaremos la vista en nuestro *mock-up* provisto en la sección de diseño, donde identificamos que cada una de las preguntas será mostrada de forma independiente. Una vez la pregunta es completada, navegaremos a la siguiente pregunta. Además, por cada pregunta añadimos una sección que nos indicará la importancia que tiene para el usuario la respuesta que la otra persona dé a la misma pregunta. Esto nos proporcionará la herramienta en nuestra próxima implementación de la búsqueda de compatibilidad para la creación de nuevas citas.

Por último, será necesario definir los diferentes servicios que necesitaremos para completar la funcionalidad. Primero, implementaremos un servicio que nos proporcione toda la información relativa a las preguntas. En él obtendremos cada una de las preguntas que hemos incluido en nuestro sistema previamente, con un listado de las posibles respuestas asociadas a esa pregunta.

A continuación, necesitaremos de un servicio que nos permita almacenar las respuestas que el usuario da a cada una de las preguntas. Este servicio será ejecutado para cada una de las

respuestas dadas, no siendo necesario, así, completar todo el formulario pudiendo posponerlo tantas veces el usuario desee.

Tras esto, y debido a que el usuario tendrá la habilidad de modificar cualquiera de las respuestas proporcionadas, necesitaremos de un servicio que actualice la información en nuestra base de datos.

Por último, en el momento de la carga de la vista debemos identificar si el usuario ha proporcionado previamente la respuesta a alguna de las preguntas y cargarla en nuestro sistema. Para ello, crearemos un servicio paralelo al que obtiene toda la información sobre las preguntas y respuestas. La idea será que cada una de las respuestas contendrá una variable que nos indicará si el usuario ha seleccionado esa respuesta.

9.7.2.4. Nueva cita

Para la implementación de este último bloque identificamos nuevamente una vista existente en la plantilla que nos servirá de base de desarrollo. Para el acceso a la creación de una nueva cita, ya que es el módulo principal de nuestra aplicación móvil, insertaremos un botón en cada una de las vistas que nos conducirá al formulario de creación de la cita.

De nuevo comenzaremos con la modificación de la vista para que se ajuste al diseño del *mock-up*. Un elemento a mencionar aquí es que nuestra vista incluirá como parte de ella un listado de los restaurantes disponibles para la cita. En este caso, previamente dada una fecha y una localización, buscaremos de entre los restaurantes suscritos los que se ajusten a la localización e incluyan disponibilidad publicada para la fecha dada. En lugar de mostrar los resultados en formato de lista, serán cargados en carrusel, permitiendo al usuario navegar entre las diferentes posibilidades simplemente deslizando el dedo para avanzar o retroceder en el listado.

Además, será posible acceder a los detalles del restaurante pulsando sobre el que se encuentre seleccionado. Esto permitirá al usuario comprobar la información y el menú que el restaurante ofrece antes de tomar la decisión. En la Figura 67 observamos la vista previamente existente a nuestro desarrollo, y el resultado de ésta una vez hemos implementado nuestras funcionalidades.

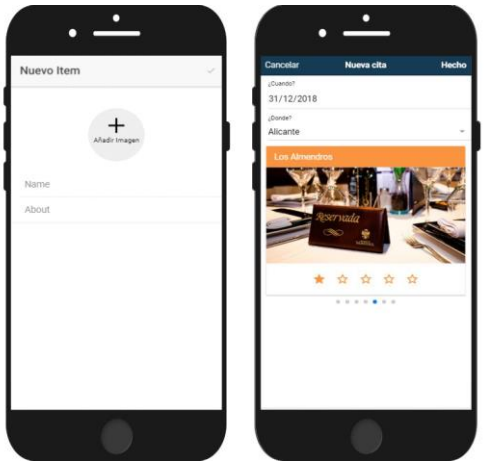

Figura 67. Sprint 6: Implementaciones de la vista de nueva cita (Fuente propia)

Será necesaria la implementación de los servicios que generen la cita. El primero de los servicios será el encargado de localizar el usuario con el que tendrá lugar la cita en nuestro sistema. Este servicio contemplará una carga de lógica importante, ya que deberá identificar cada uno de las preguntas, respuestas proporcionadas por los usuarios y descartes en función de las coincidencias entre ellas. Una visual rápida sobre la funcionalidad será: si el usuario generando la cita es un hombre, y en las preferencias sexuales nos ha indicado que es heterosexual, deberemos entonces descartar a todos los hombres y mujeres con preferencia por las mujeres de posibles candidatos; si, además, nuestro usuario ha indicado que no fuma y que la respuesta del otro usuario debe coincidir (es decir, que el otro usuario tampoco fume), todas las personas que hayan respondido “si” para la pregunta en cuestión serán descartados.

La lógica de gestión de compatibilidad tendrá que ser dividida entre la base de datos y nuestra API para lograr un tiempo de respuesta apropiado a la operación. Esto se realizará de forma que la base de datos, mediante instrucciones como "INNER JOIN” se encargará de descartar a los usuarios que queden automáticamente descartados por incompatibilidad de respuestas o localización. Nuestra API recibirá entonces una lista más acotada de posibles candidatos, sobre los que será necesario iterar para darle a cada uno de ellos un valor de compatibilidad basado en la medida de importancia que el usuario indica al completar su formulario para cada una de las preguntas. Por último, la cita será generada con el usuario de mayor porcentaje de compatibilidad, comprobando previamente si existe una cita anterior con ese usuario. En caso de que exista realizaremos la comprobación con el siguiente usuario de la lista, y así sucesivamente hasta identificar el candidato apropiado.

Para concluir el proceso, la referencia del usuario compatible será enviada, junto con el resto de detalles de la cita, a un segundo servicio que será el encargado de insertar la información sobre la nueva cita en nuestro sistema.

9.7.3. Pruebas y validación

En esta ocasión, ya que nos encontramos en el último *sprint*, planificamos la validación de la aplicación móvil al completo. Definiremos, para ello, la serie de casos de prueba para cada uno de los componentes desarrollados.

9.7.3.1. Login

Para la validación de esta funcionalidad reutilizaremos los casos de prueba definidos para la aplicación de gestión. En los resultados podemos observar de nuevo los mensajes de alerta cuando la información introducida no es correcta. Supera con éxito las pruebas.

9.7.3.2. Registro

Para la validación de esta funcionalidad reutilizaremos los casos de prueba definidos para la aplicación de gestión. En los resultados podemos observar de nuevo los mensajes de alerta cuando la información introducida no es correcta. Supera con éxito las pruebas.

9.7.3.3. Configuración

Para esta sección tendremos que asegurar que la navegación dentro del sub-menú creado para la vista de configuración funciona de manera correcta. Supera con éxito las pruebas realizadas.

9.7.3.4. Configuración: Gestionar usuarios bloqueados

Para esta sección será necesario realizar la validación sobre la información de bloqueo introducida. En primer lugar, nuestra vista se encuentra libre de información:

1. Introduciremos un dato diferente a un número de teléfono para realizar el bloqueo del usuario. Esta opción no debería encontrarse disponible en el sistema.
2. Introduciremos un número de teléfono para realizar el bloqueo. La lista de bloqueos debería actualizarse de forma automática para incluir el nuevo bloqueo.
3. Eliminaremos el bloqueo generado. La lista de bloqueos debería actualizarse de forma automática descartando la información bloqueada.

Supera las pruebas con éxito.

9.7.3.5. Configuración: Cambiar contraseña

Para la validación de esta funcionalidad reutilizaremos los casos de prueba definidos para la aplicación de gestión. En los resultados podemos observar de nuevo los mensajes de alerta cuando la información introducida no es correcta. Supera con éxito las pruebas.

9.7.3.6. Configuración: Modificar datos de la cuenta

Para la validación de esta funcionalidad reutilizaremos los casos de prueba definidos para la aplicación de gestión. En los resultados podemos observar de nuevo los mensajes de alerta cuando la información introducida no es correcta. Supera con éxito las pruebas.

9.7.3.7. Configuración: Cancelar cuenta

Para la validación de esta funcionalidad reutilizaremos los casos de prueba definidos para la aplicación de gestión. Supera con éxito las pruebas.

9.7.3.8. Listado de citas

Para la validación de esta funcionalidad será necesario iterar sobre las diferentes citas listadas, el bloque al que pertenecen y los detalles que cada una de ellas presenta en la vista de detalle:

1. Incluiremos en nuestro sistema un listado de citas variando entre fechas anteriores y posteriores a la fecha actual, siendo el estado para las anteriores el de cita completada, y variando el estado para las citas posteriores entre aceptada y por confirmar. Además, para las citas anteriores definiremos la valoración del usuario para una de ellas.
2. Comprobaremos que cada una de las citas se lista en la sección correspondiente: próximas citas o citas anteriores.
3. Para las citas anteriores, comprobaremos que para la que hemos definido previamente la valoración, esta funcionalidad se encuentra descartada.
4. Para las citas actuales, comprobaremos que las citas por confirmar nos permiten iterar sobre los detalles de ella para modificarla y mandar la nueva solicitud al usuario candidato.
5. Para las citas actuales, comprobaremos que las citas aceptadas no nos permiten modificar los datos, y la única funcionalidad activa es la de cancelación. Para la

cancelación de la cita reproduciremos los casos de prueba definidos en el *back-office* donde implementábamos anteriormente esta funcionalidad.

Supera las pruebas con éxito.

9.7.3.9. Listado de restaurantes

Para la validación de este módulo será necesario iterar sobre el listado de restaurantes ofrecido, el funcionamiento del filtro de búsqueda y los detalles proporcionados para cada restaurante.

1. Comprobamos que en primera instancia todos los restaurantes del sistema se encuentran listados al cargar la vista.
2. Comprobamos que introducir en el campo de búsqueda valor del tipo nombre, ciudad o tipo el filtro se realiza de forma correcta.
3. Comprobamos que en la vista de detalle del restaurante, la información mostrada pertenece directamente al restaurante en cuestión.

Supera las pruebas con éxito.

9.7.3.10. Formulario

Para la validación de este módulo debemos iterar sobre la inclusión de todas las preguntas y respuestas que tenemos definidas en nuestro sistema en la vista, y que la información introducida por el usuario se guarda y carga de forma correcta.

1. Comprobaremos que en la navegación de las preguntas se incluyen todas las preguntas que se encuentran en el sistema, y que para cada una de ellas se incluyen las respuestas correspondientes.
2. Comprobaremos que la información del usuario se almacena en la base de datos una vez responda a cada una de las preguntas.
3. Comprobaremos que al volver a acceder a la vista del formulario, la información previamente introducida por el usuario se encuentra disponible.

Supera las pruebas con éxito.

9.7.3.11. Nueva cita

Para la validación de este módulo debemos iterar sobre la inclusión de todas las localizaciones donde haya un restaurante registrado en el sistema, que los restaurantes listados como posible anfitrión se encuentren en la ciudad y tengan disponibilidad para el día en cuestión, y que el candidato seleccionado por el sistema es compatible basándonos en las respuestas proporcionadas por ambos.

1. Completaremos el formulario con una serie de usuarios con el fin de realizar la comprobación sobre descartes de candidatos.

2. Comprobaremos que al acceder en la vista de nueva cita, las únicas ciudades listadas son las que contienen restaurantes registrados en la localización. Realizaremos modificaciones sobre los restaurantes existentes para comprobar el cambio de información en la vista.
3. Comprobaremos que los restaurantes listados, además, tengan disponibilidad publicada para el día seleccionado. Realizaremos modificaciones sobre la disponibilidad para comprobar el cambio de información en la vista.
4. Comprobaremos en varias iteraciones que los usuarios para la cita generada no son descartables, es decir, que no contienen una incompatibilidad directa en las respuestas proporcionadas.

Supera las pruebas con éxito.

10. Conclusiones y trabajo futuro

Tras varios años de realización de diversos proyectos a lo largo de la titulación, el TFG es, sin duda, el más complejo a afrontar. En él, hemos necesitado aplicar todos los conocimientos adquiridos a través de las diferentes asignaturas para construir el esqueleto y poder darle forma.

La primera reflexión que nos viene a la mente es lo duro que es afrontar un proyecto de estas dimensiones en solitario. La toma de decisiones y la visión de mercado y posibilidades se torna más complicada cuando no existe un equipo de trabajo. En los primeros pasos del proyecto definíamos qué era nuestro proyecto, qué buscábamos con nuestro desarrollo y cuáles eran, a día de hoy, las tendencias en el mercado relacionadas. Es muy complicado no ser subjetivos en un punto tan trivial, donde nuestra idea sobre el proyecto puede no ser la más realista. Aquí es donde un trabajo en equipo juega un papel fundamental, poniendo en la mesa todos los pros y los contras, construyendo en común.

Derivado directamente del trabajo unipersonal, otro de los grandes retos en este proyecto es la necesidad de haber adoptado tan diversos roles a lo largo de su desarrollo. Como ya hemos mencionado, hemos necesitado realizar una primera labor de análisis, estudio de mercado y toma de decisiones. Pero, además, durante las siguientes fases se ha hecho necesario trabajar en todos los campos que abarca un proyecto software. Gestión de servidores, implementación y validación son campos en los que hemos trabajado a lo largo de estos años, y que, en este proyecto, hemos necesitado poner en común y trabajar duro para cumplir nuestros objetivos.

Al concluir BlinDates hemos obtenido un sistema que permitirá, a los usuarios que den uso de él, buscar nuevas experiencias en el campo de las citas a ciegas. Además, buscamos que los restaurantes abran una nueva vía de negocio y marketing al suscribirse en nuestra aplicación y ofrecerse como anfitriones para las citas mencionadas.

Como todo proyecto personal, nunca encontraremos el proyecto concluido, no en términos de funciones o elementos a añadir, si no en cuanto a las posibles mejoras dentro de lo existente. Incluso en el resultado que exponemos finalmente, hemos iterado en alguna ocasión sobre lo ya implementado incluyendo nuevos conocimientos que hemos ido adquiriendo externamente y que nos han hecho replantearnos la forma en la que habíamos procedido.

El proyecto está sin duda inacabado. Es una versión prototipo de un sistema complejo y, probablemente, en constante evolución. El desarrollo actual compone los elementos básicos necesarios para una primera versión funcional, pero sobre la que necesitaremos iterar para cumplir los estándares de las aplicaciones que encontramos a día de hoy en el mercado. Aunque habiendo cumplido los objetivos que nos proponíamos al comienzo del proyecto, estos contemplaban un pequeño porcentaje del trabajo real que supondría.

A continuación, detallaremos alguno de los elementos sobre los que creemos que es imprescindible hacer foco en la continuación del proyecto:

10.1. Social *login*

Este es sin duda uno de los elementos más estandarizados en las aplicaciones a día de hoy. Ya sea web o móvil, permitir que un usuario acceda directamente mediante la autenticación de otra plataforma repercute de forma positiva. De forma general, uno de los beneficios es sin duda que el usuario confiará más en una aplicación interconectada con las aplicaciones que normalmente utilizan (twitter, Facebook, google…). Como punto más específico para nuestra aplicación, nos permitirá que el usuario señale los bloqueos a usuarios para la generación de las citas a partir de sus contactos en las otras redes sociales, haciendo de esta funcionalidad algo mucho más usable y visual.

10.2. Geolocalización

Otro de los imprescindibles en aplicaciones donde se tratan elementos de espacio. La geolocalización implementada en nuestra aplicación móvil nos permitirá agilizar los procesos tanto de búsqueda de restaurantes como el filtrado de candidatos en la generación de nuevas citas.

10.3. Estudio social sobre la compatibilidad entre usuarios

Un punto clave para nuestra aplicación. En este momento el formulario que presentamos a los usuarios contiene una serie de preguntas que hemos extraído a partir de los cuestionarios que otras plataformas ofrecen. Pero, para destacar como aplicación y obtener una satisfacción real del usuario en cada cita, será necesario realizar un estudio social sobre las variables que definen la compatibilidad entre usuarios y proporcionar un listado de preguntas clave en nuestro formulario.

10.4. Sistema de notificaciones y correo electrónico

Elementos clave en la fidelización de usuarios. La implementación de notificaciones en nuestro móvil alertarán al usuario de las nuevas citas o cambios de estado que estas sufran. Además, recordatorios como completar el formulario o cita en el día de hoy ayudarán a que el usuario acceda casi diariamente a nuestra aplicación.

El correo electrónico, aunque más anticuado, sigue siendo un sistema implantado en las aplicaciones como canal de comunicación para confirmación de eventos, como registro o cambio de contraseña, o mensajes informativos relativos a términos y condiciones de uso o nuevas funcionalidades en el sistema.

10.5. Implementación de entornos de desarrollo

De carácter interno, este es un punto necesario para posteriores desarrollos del producto. Teniendo en mente una continuación en la implementación, será imprescindible contar con los diferentes entornos que define el proceso. La separación del código nuevo desarrollado, el código dispuesto para su validación y el código que actualmente se encuentra en uso nos proporcionará la herramienta para que el trabajo de las diferentes áreas funcione correctamente.

Referencias

1. Juan Carlos Mejía Llano. Estadísticas de redes sociales 2018. Disponible en: http://www.juancmejia.com/marketing-digital/estadisticas-de-redes-sociales-usuarios-de-facebook-instagram-linkedin-twitter-whatsapp-y-otros-infografia/#Informe_detallado_usuarios_redes_sociales_WeAreSocial_y_Hootsuite
2. Dra Cristina Miguel Martos. Universidad de Leeds. Disponible en: http://media.leeds.ac.uk/people/cristina-miguel-martos/
3. Natalia Marcos. "En las tripas de First Dates". Disponible en: https://elpais.com/cultura/2016/12/16/television/1481896996_339439.html
4. José Antonio Molina del Peral, psicólogo experto en adicciones y director del centro Psicohealth. Disponible en: http://www.psicologo-terapeuta.com/
5. Lean Canvas en nuevos proyectos. Disponible en https://javiermegias.com/blog/2012/10/lean-canvas-lienzo-de-modelos-de-negocio-para-startups-emprendedores/
6. Daniel Aparicio. Artículo sobre ligar por internet. Disponible en: https://www.20minutos.es/noticia/1873416/0/apps/ligar/encontrar-pareja/
7. Virginia Collera. Ligar en tiempos modernos. Disponible en: https://elpais.com/elpais/2015/10/23/eps/1445602424_708600.html
8. Carmen Jané. Estudio estadístico sobre las aplicaciones de ligue entre los jóvenes. Disponible en: https://www.elperiodico.com/es/sociedad/20170429/las-apps-de-ligue-triunfan-entre-los-jovenes-6006210
9. Tinder. Red Social. En forma de página web y aplicación móvil. Disponible en: https://play.google.com/store/apps/details?id=com.tinder&hl=es
10. Happn. Red Social. En forma de aplicación móvil. Disponible en: https://www.happn.com/es/
11. Badoo. Red Social. En forma de página web y aplicación móvil. Disponible en: https://badoo.com/es/
12. Meetic. Red Social. En forma de página web y aplicación móvil Disponible en: https://www.meetic.es/
13. Grindr. Red Social. En forma de aplicación móvil. Disponible en: https://www.grindr.com/
14. OkCupid, página web de citas y lanzadora de Crazy Blind Dates. Disponible en: https://www.okcupid.com/

15. Metodología SCRUM. Disponible en https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html
16. 1&1MiWeb. Gestión de servidores y creación de página web. Disponible en: https://www.1and1.es/
17. Hostalia. Gestión de servidores. Disponible en: https://www.hostalia.com/
18. Hostinguer. Gestión de servidores. Disponible en: https://www.hostinger.es/
19. Amazon Server. Servicios de informática en la nube. Disponible en: https://aws.amazon.com/es/
20. Amazon Route 53. Servicio de DNS. Disponible en: https://aws.amazon.com/es/route53/
21. Amazon Simple Email Service. Servicio de correo web. Disponible en: https://aws.amazon.com/es/ses/
22. Mongo DB. Servicio de Base de datos. Disponible en: https://www.mongodb.com/
23. Oracle. Servicio de Base de datos. Disponible en: https://www.oracle.com/es/corporate/pricing/index.html
24. Lista de precios por licencia de uso de Oracle. Disponible en: http://www.oracle.com/us/corporate/pricing/applications-price-list-070574.pdf
25. SQL Server. Servicio de base de datos. Disponible en: https://www.microsoft.com/es-es/sql-server/
26. MySQL. Servicio de base de datos. Disponible en: https://www.mysql.com/
27. NodeJS. Lenguaje de programación del lado de servidor. Basado en JavaScript. Disponible en: https://nodejs.org/es/
28. Ruby on Rails. Lenguaje de programación web. Disponible en http://rubyonrails.org.es
29. Python. Lenguaje de scripting. Disponible en: https://www.python.org/
30. PHP. Lenguaje de programación del lado de servidor. Disponible en: http://php.net/
31. Symfony. Framework de PHP. Disponible en: http://symfony.es/
32. Single Page application. Paradigma de desarrollo web. Disponible en: https://es.wikipedia.org/wiki/Single-page_application
33. JavaScript. Lenguaje de desarrollo web. Disponible en: https://www.javascript.com/
34. ReactJS. Biblioteca de JavaScript. Disponible en: https://github.com/reactjs
35. EmberJS. Framework de JavaScript. Disponible en: https://www.emberjs.com/
36. AngularJS. Framework de JavaScript. Disponible en: https://angularjs.org/
37. Phonegap. Librerías de desarrollo de aplicaciones híbridas. Disponible en: http://phonegap.com/

38. Ionic. Framework de desarrollo de aplicaciones multiplataforma. Disponible en: https://ionicframework.com/
39. Subversion. Sistema de control de versiones. Disponible en: https://subversion.apache.org/
40. Mercurial. Sistema de control de versiones. Disponible en: https://www.mercurial-scm.org/
41. Git. Sistema de control de versiones. Disponible en: https://git-scm.com/
42. GitLab. Herramienta para la gestión de repositorios Git. Disponible en: https://about.gitlab.com/
43. Artículo sobe la implementación de *sprints* en la herramienta de Team Foundation Server. Disponible en: https://msdn.microsoft.com/es-es/library/ms181692(v=vs.120).aspx
44. Artículo sobre la estimación ágil con la técnica Planning Poker. Disponible en: https://samuelcasanova.com/2016/01/estimacion-agil-con-la-tecnica-planning-poker/
45. Guía para la creación de una instancia en Amazon. Disponible en: https://docs.aws.amazon.com/es_es/efs/latest/ug/gs-step-one-create-ec2-resources.html
46. Guía sobre las direcciones IP elásticas de Amazon. Disponible en: https://docs.aws.amazon.com/es_es/AWSEC2/latest/WindowsGuide/elastic-ip-addresses-eip.html
47. Guía sobre la configuración de dominios y subdominios en Route 53, Amazon. Disponible en; https://aws.amazon.com/es/premiumsupport/knowledge-center/create-subdomain-route-53/
48. Tutorial de Symfony 3. Disponible en: https://www.mercenariophp.com/tutorial-symfony/
49. Ng-src. Directiva de AngularJS para el manejo dinámico de imágenes. Disponible en: https://docs.angularjs.org/api/ng/directive/ngSrc
50. FileReader. Librería de manejo de archivos. Disponible en: https://developer.mozilla.org/es/docs/Web/API/FileReader
51. Guía de AngularJS. Elimincación de # en la url. Disponible en: https://docs.angularjs.org/guide/$location#html5-mode
52. Ng-if. Directiva de AngularJS para la aplicación de condiciones sobre los elementos directamente en la vista. Disponible en: https://docs.angularjs.org/api/ng/directive/ngIf
53. Propagación de eventos en AngularJS. Disponible en: https://www.uno-de-piera.com/eventos-en-angularjs/
54. Documentación de Ionic. Implementando una aplicación desde cero. Disponible en: https://medium.com/@nestorjerez/desarrolla-tu-primera-aplicaci%C3%B3n-m%C3%B3vil-con-ionic-3-desde-0-5056928152da
55. Almacenamiento de datos en Ionic. Disponible en: https://ionicframework.com/docs/storage/

Apéndice I

Podemos observar en la Figura 68 el listado de elementos de trabajo definidos al comienzo del proyecto. TFS nos permitirá crear tantos elementos de trabajo como sean necesarios para que más adelante sean introducidos en su correspondiente ciclo de desarrollo.

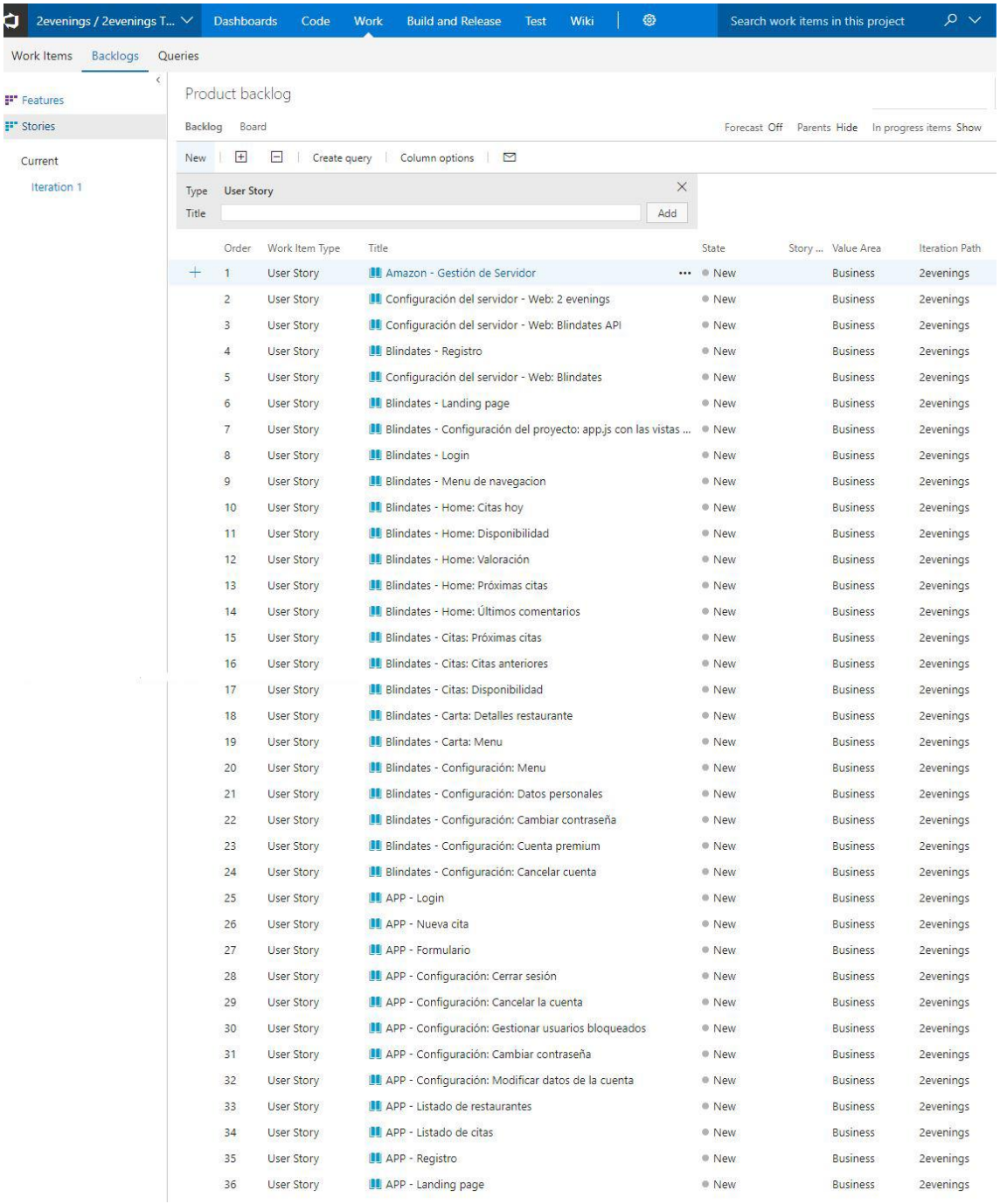

Figura 68. Identificación de elementos de trabajo: visión general
(Fuente propia)

Apéndice II

En la Figura 69 observamos los elementos de trabajo que hemos incluido para el primer *sprint*, una vez la sesión de análisis y estimaciones ha concluido. Quedan, así, especificados para cada uno de los *stories* los elementos sobre los que trabajaremos y el tiempo que estimamos que nos llevará completar la tarea.

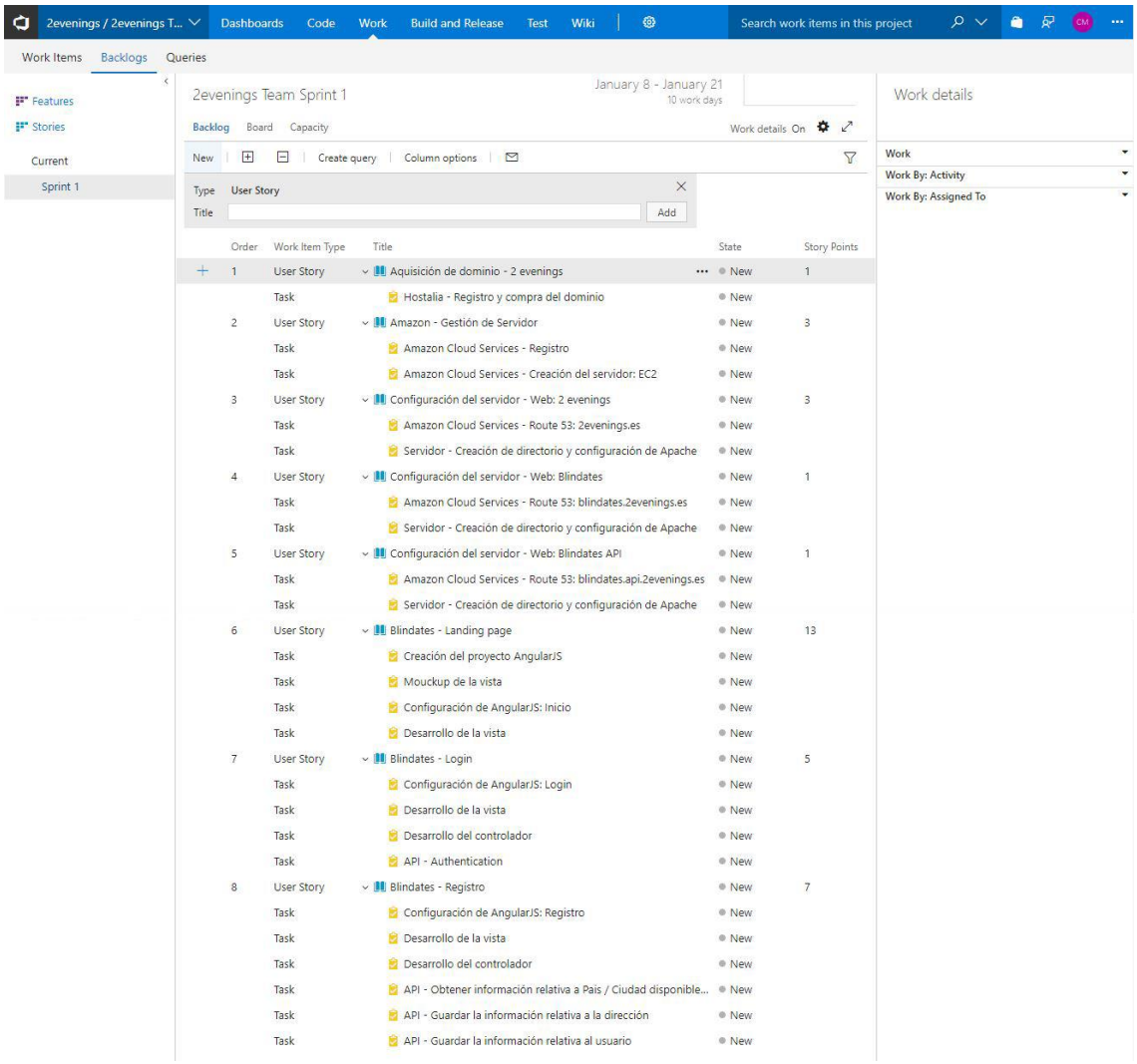

Figura 69. Sprint 1: Stories analizados y estimados
(Fuente propia)

Apéndice III

En la Figura 70 observamos los elementos de trabajo que hemos incluido para el segundo *sprint*, una vez la sesión de análisis y estimaciones ha concluido. Quedan, así, especificados para cada uno de los *stories* los elementos sobre los que trabajaremos y el tiempo que estimamos que nos llevará completar la tarea.

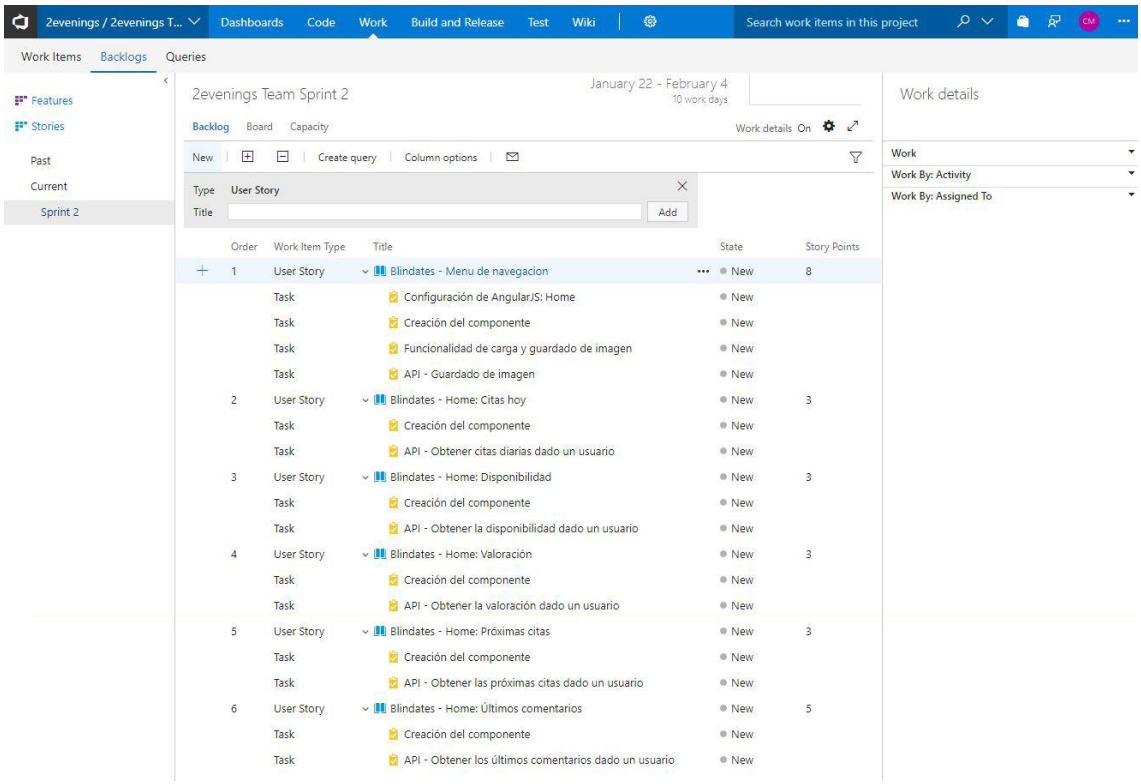

Figura 70. Sprint 2: Stories analizados y estimados
(Fuente propia)

Apéndice IV

En la Figura 71 observamos los elementos de trabajo que hemos incluido para el tercer *sprint*, una vez la sesión de análisis y estimaciones ha concluido. Quedan, así, especificados para cada uno de los *stories* los elementos sobre los que trabajaremos y el tiempo que estimamos que nos llevará completar la tarea.

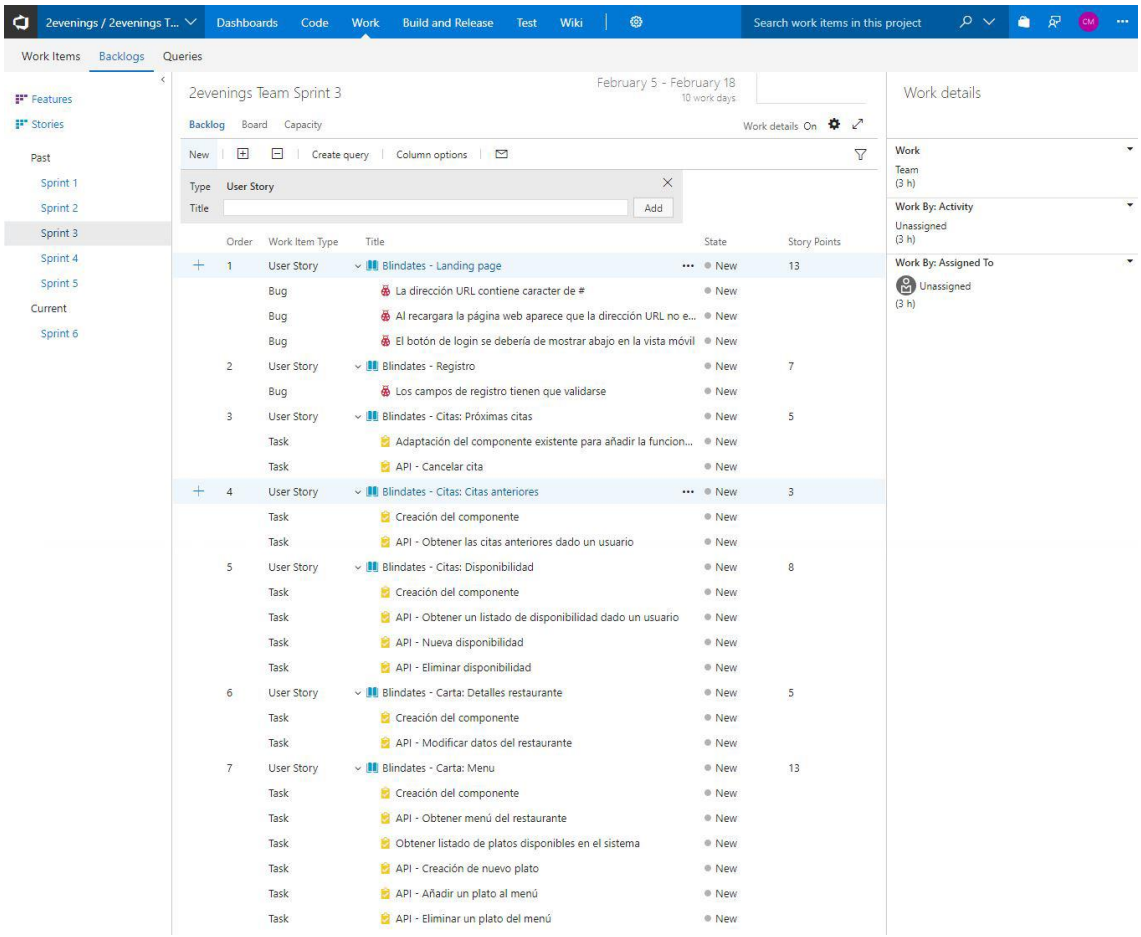

Figura 71. Sprint 3: Stories analizados y estimados
(Fuente propia)

Apéndice V

En la Figura 72 observamos los elementos de trabajo que hemos incluido para el cuarto *sprint*, una vez la sesión de análisis y estimaciones ha concluido. Quedan, así, especificados para cada uno de los *stories* los elementos sobre los que trabajaremos y el tiempo que estimamos que nos llevará completar la tarea.

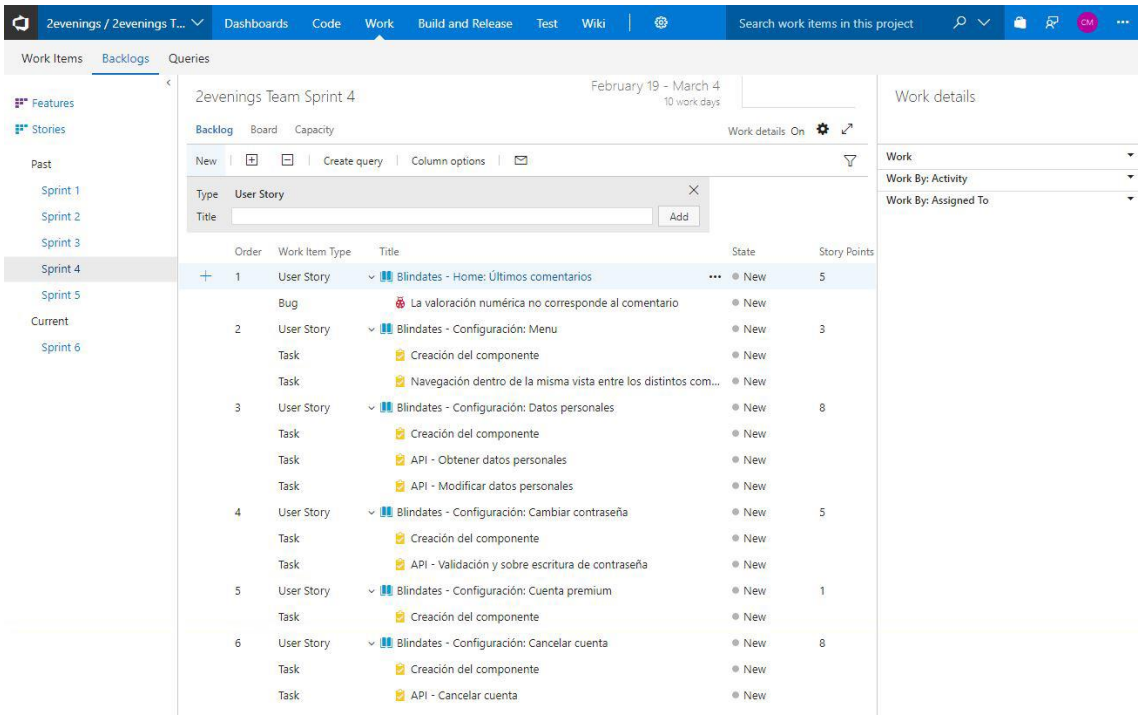

Figura 72. Sprint 4: Stories analizados y estimados
(Fuente propia)

Apéndice VI

En la Figura 73 observamos los elementos de trabajo que hemos incluido para el quinto *sprint*, una vez la sesión de análisis y estimaciones ha concluido. Quedan, así, especificados para cada uno de los *stories* los elementos sobre los que trabajaremos y el tiempo que estimamos que nos llevará completar la tarea.

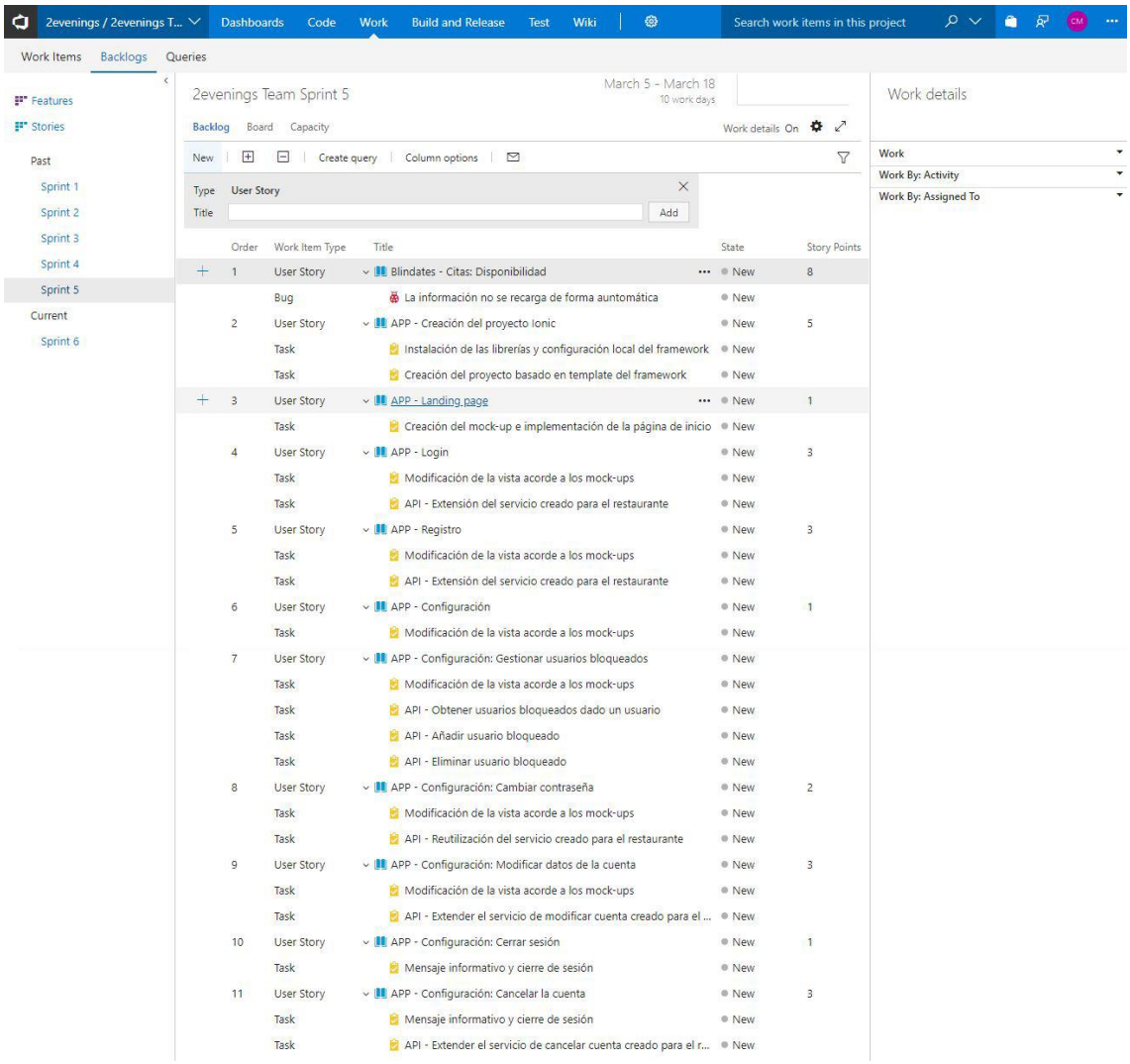

Figura 73. Sprint 5: Stories analizados y estimados
(Fuente propia)

Apéndice VII

En la Figura 74 observamos los elementos de trabajo que hemos incluido para el sexto *sprint*, una vez la sesión de análisis y estimaciones ha concluido. Quedan, así, especificados para cada uno de los *stories* los elementos sobre los que trabajaremos y el tiempo que estimamos que nos llevará completar la tarea.

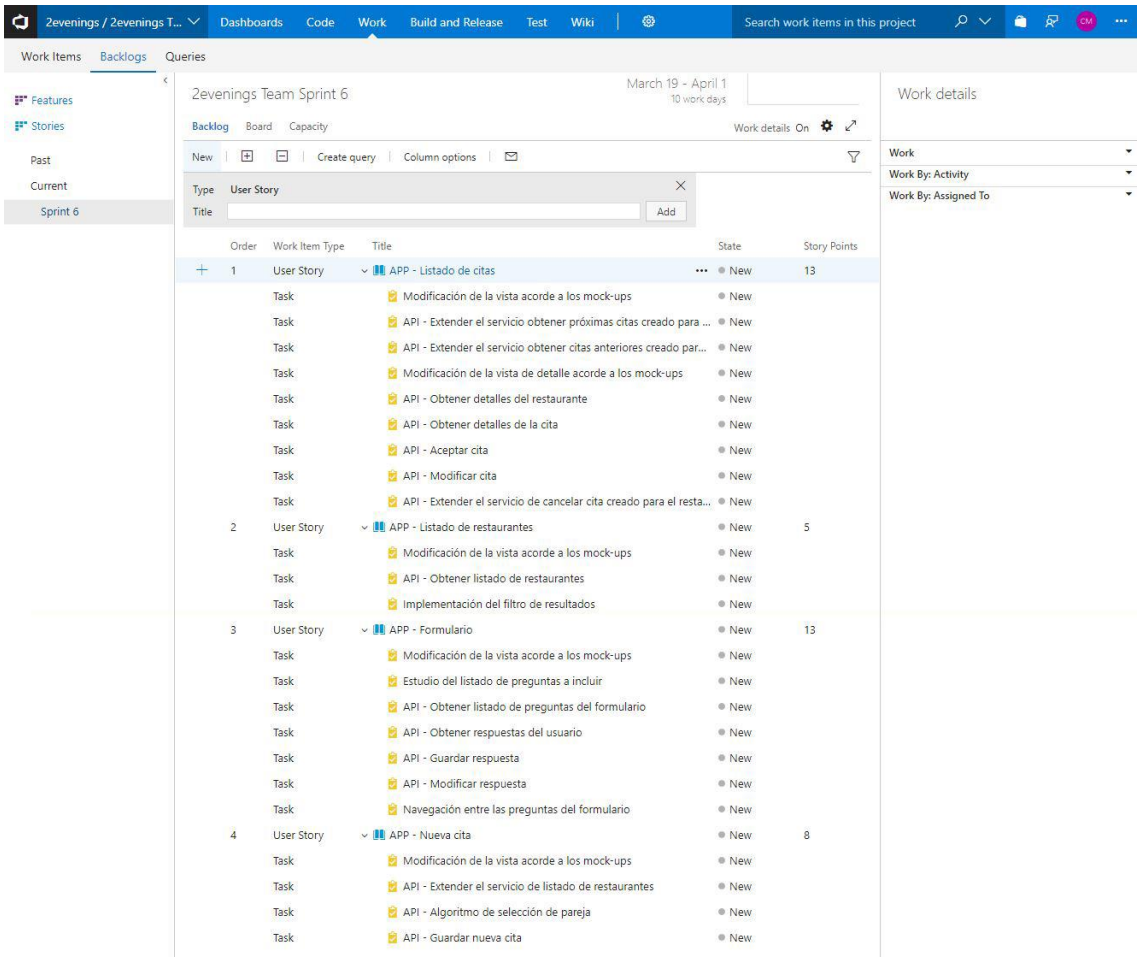

Figura 74. Sprint 6: Stories analizados y estimados
(Fuente propia)