PROYECTO FIN DE CARRERA

Memoria del proyecto

Título: Diseño e implementación de un micromercado virtual, el de los profesionales técnicos del hogar

Autor: Mario Catalán Nordbeck

Fecha: 22 de septiembre de 2016

Director: Enric Mayol Sarroca

Departamento del director: ESSI

Centro: Facultad de Informática de Barcelona (FIB)

Universidad: Universidad Politécnica de Cataluña (UPC)

Índice

1. Introducción

1.1. Definición de la aplicación

Esta aplicación brindará a oferentes (profesionales técnicos del hogar) y demandantes (consumidores de los servicios ofrecidos por los profesionales técnicos del hogar, en general reformas, reparaciones y renovación) un micromercado virtual en el que encontrarse, en el que comunicarse y en el que realizar transacciones financieras.

Evidentemente, estos tres acontecimientos se realizarán en secuencia: primero el encuentro, luego la comunicación, y finalmente la transacción. Ni qué decir tiene que la ocurrencia de uno de estos acontecimientos no implica la necesaria ocurrencia del acontecimiento que le sucede. Sin embargo, cuando se dan los tres acontecimientos, y se dan ordenados, denominamos este suceso como proceso completo. En consecuencia, un proceso incompleto es uno en el que oferente y demandante se encuentran pero no establecen comunicación, o se comunican pero no consuman la transacción. Si oferente y demandante no se encuentran, entonces no existe tal proceso: el proceso se inicia en el encuentro.

Nuestra aplicación se vale del principio de localidad para procurar los encuentros. Así por ejemplo a un demandante dado se le sugerirán encuentros con los oferentes más cercanos. Esto no quiere decir que dichos oferentes estén cerca del demandante, sino simplemente que son los que están más cerca.

Las motivaciones para hacer uso del principio de localidad son varias y claras, y la mayoría de estas motivaciones las comparten en general las economías de cercanías. Una de estas

motivaciones es que estos servicios que realizan los profesionales técnicos del hogar se realizan in situ, esto es, en el lugar. Para hacer el presupuesto, probablemente la mejor manera es desplazarse hasta el lugar donde se debe realizar el trabajo. Los presupuestos por lo general son gratuitos, de modo que la motivación para desplazarse hasta el lugar que precisa obra y presupuesto disminuye a medida que la distancia hasta el lugar aumenta. Además, si el trabajo se prolonga durante varios días el coste en desplazamientos tanto en tiempo como en dinero aumenta considerablemente, lo cual resulta desmotivador para el cliente o demandante. Al final lo que se busca es la inmediatez, la inmediatez al encontrarse, la inmediatez al contactar y la inmediatez al presupuestar, al aceptar el presupuesto, al comenzar la obra, y al terminar la obra. Las distancias cortas siempre son más inmediatas: por ejemplo, Juan, fontanero, no tendrá problemas en subir un momento a visitar a su vecino Pedro del quinto para hacerle un presupuesto sobre el coste de la reparación de una tubería que pierde agua.

Sigamos con más inmediatez: inmediatez en la comunicación. Para comunicarse lo más rápido que hay hoy día es la mensajería instantánea, más rápido incluso que llamar. Además, no solamente es la forma más rápida e inmediata de comunicación sino que también es la preferida por la mayoría de las personas, probablemente debido a que es menos directa y por tanto menos agresiva que la llamada. Sin duda es una gran ventaja poder contactar con un obrero en segundos, y acordar su visita a nuestro piso para realizar el presupuesto de obra en menos de cinco minutos. Todo esto sin más esfuerzo que intercambiar menos de una decena de mensajes por cabeza.

La tercera y última inmediatez que vamos a tratar es la inmediatez en los pagos. Pulsar un botón, introducir los credenciales de PayPal, y transacción realizada. Todo ello sin tener que entrar en nuestro espacio personal dentro de la web del banco, seleccionar una de nuestras

cuentas, introducir todos los detalles de la transacción, lo cual es un proceso fatigoso y muy dado a errores, especialmente a la hora de introducir el número de la cuenta de destino, y finalmente superar los varios desafíos de seguridad que nos plantea el banco, como introducir el código que te han enviado al teléfono móvil por SMS o introducir el código que corresponde a otro código en un listado de códigos impreso en una tarjeta física facilitada en su momento por el propio banco para permitir y autorizar los pagos en línea.

Entregar efectivo no es tan costoso. Primero hay que disponer de él, y si no se tiene ir a un cajero que nos permita sacar dinero de nuestra cuenta, tarjeta o libreta mediante. El desplazamiento desde casa hasta la sucursal y desde la sucursal hasta casa probablemente sea para muchos todavía más exigente que comprobar que el código IBAN de la cuenta de destino es correcto, que la cantidad tiene bien colocada la coma decimal, y cuál es el código de seis cifras que me acaban de enviar al teléfono inteligente para después introducirlo en el formulario correspondiente. Luego está el proceso de contar billetes, donde el pagador comprueba y recomprueba que no esté pagando de más ni tampoco de menos, y donde el pagado comprueba que no le estén pagando de menos. Es probable que este conteo, al ser de billetes físicos, requiera de un esfuerzo equiparable o incluso superior al necesario para comprobar que los datos de transferencia introducidos en la sucursal virtual del banco son correctos y para copiar códigos de, a lo sumo, seis dígitos. Dicha probabilidad aumenta con el número de billetes de que consta el pago.

En cualquier caso, queda claro que para el cliente o demandante lo más cómodo es pagar a través de la aplicación. Sin embargo, igual de claro está que al profesional no le va a interesar que se le deduzcan del cobro la tasa de PayPal y la tasa de nuestro micromercado virtual si no se le da nada a cambio. De modo que se le dará algo: reputación en nuestro micromercado. La

reputación es en realidad algo más complejo, pero sin duda en este caso uno de sus principales componentes es la cantidad de dinero ingresado por trabajos u obras realizados. Sin duda, y ya que puede elegir, el cliente o demandante se pondrá en contacto con los profesionales u oferentes más reputados de su zona, esto es, con los más reputados de entre sus más cercanos. De este modo, a los profesionales les interesará, al menos, ser los más reputados de su zona. Esto introduce desafíos adicionales que serán tratados a su debido tiempo, como por ejemplo la existencia de barreras de entrada para los profesionales de reciente ingreso en el micromercado virtual.

1.2. Motivación

La motivación de este proyecto es de índole revolucionaria-experimental. Revolucionaria para romper con el modelo establecido en este tipo de aplicaciones, puesto que desde hace años están basadas en dificultar la comunicación directa entre oferentes y demandantes (varias de estas aplicaciones no se dedica en exclusiva a los servicios técnicos del hogar), en modelos del dominio extraordinariamente profundos y complejos, cuyo valor asciende a millones de dólares, que se traducen en formularios extensos y tediosos para el cliente, donde es el oferente el que busca solicitudes de servicio publicadas por demandantes (en general, cercanos), y donde el oferente debe pagar para poder optar a la realización del servicio, por ejemplo obteniendo acceso a los datos de contacto del demandante, si bien en ningún caso tiene garantizado ser elegido.

En primer lugar, dificultar la comunicación directa entre oferentes y demandantes es crítico para estas aplicaciones, puesto que obtienen dinero habilitando precisamente esta comunicación a

cambio de un módico precio que se le cobra al oferente. Ese es su modelo de negocio, y si lo cambiaran perderían su actual esencia para terminar convertidos en otra cosa.

En segundo lugar, los modelos del dominio de extraordinaria profundidad y complejidad son el principal activo de estas empresas, del mismo modo que para las casas de apuestas deportivas su mayor activo son sus algoritmos de predicción de resultados.

Estos modelos tan complejos y profundos se traducen necesariamente en formularios más extensos y difíciles para quienes los rellenan, en este caso los demandantes o solicitantes de servicio. Cuanto mejor es el sistema de información subyacente, lo cual puede estimarse por el valor en millones de dólares del servicio, más extensos y difíciles son dichos formularios. Esto resulta frustrante para quien desea contactar con alguien y que se encargue este alguien de los tecnicismos. Sin embargo, no todo es malo: la información recogida por estos formularios es de inmenso valor para hacer analítica de datos e inteligencia empresarial.

Cambiar el rol del buscador, que busque el profesional en vez del solicitante, tiene sus ventajas y sus inconvenientes. Por un lado, la principal ventaja es que elimina las barreras de entrada para los profesionales de reciente ingreso que existen cuando es el solicitante o demandante el que busca al profesional, y se guía de la reputación que ostenta éste último. Por otro lado, el principal inconveniente es que no permite al demandante elegir, o al menos no de la misma manera: en el mejor de los casos podrá elegir de entre un puñado de profesionales cuya reputación no está verificada por el sistema (el sistema no da garantías de que realmente se haya realizado el trabajo y el consecuente pago); en el peor caso, el demandante elegirá al primero o al único que se ponga en contacto con él.

Por último, en el modelo tradicional el profesional paga por la posibilidad de ser elegido. En la alternativa que presentamos al modelo tradicional, el profesional paga solo si es elegido efectivamente, y solo si él desea cobrar a través de la aplicación.

El mayor problema al que se enfrentan estas empresas tradicionales de contratación de servicios es que no pueden comprobar la autenticidad del solicitante. Esto da pie a que haya personas con malas intenciones que publiquen su solicitud, puesto que publicar es gratuito, den datos de contacto falsos o inexistentes, y que se desentiendan, de modo que todos los profesionales que paguen por los datos de contacto o por la posibilidad de ser elegidos para el trabajo se sentirán estafados. Algunos de estos profesionales que se sienten estafados llegan a asegurar que quienes publican estas solicitudes falsas son empleados de la propia compañía para incrementar los ingresos. En definitiva, un problema muy difícil con el que lidiar.

2. Descripción del proyecto

2.1. Detección de oportunidades

Las oportunidades siempre deben ser contextualizadas si quieren analizarse con cierto rigor.

En esta sección realizaremos la contextualización espacial y temporal de las oportunidades.

En lo que respecta a contextualización espacial, es evidente que no es lo mismo analizar las oportunidades aquí en España que en China, en India o, sobre todo, en Estados Unidos. En España la competencia más fuerte es la que presenta Chronoshare, que cuenta con un total de más de 650 mil profesionales. Además, en España existen indicios tímidos de aplicaciones que presentan un enfoque revolucionario como el que aquí presentamos, pero de momento no se ha llegado a nada. En este sentido, el hecho de que estas aplicaciones con un enfoque revolucionario todavía no hayan llegado a nada debería considerarse como una oportunidad para nosotros en el marco de la contextualización temporal.

En China lo que se detecta es nuestra debilidad: el idioma. Sin duda un mercado no accesible para nosotros. En India lo que encontramos son múltiples amenazas, en forma de aplicaciones que son competencia y de realización extraordinaria, puesto que es ampliamente conocido el nivel de los programadores indios.

Estados Unidos es el mercado de startups más competitivo del mundo. Como tal, existen múltiples servicios solamente disponibles en Estados Unidos de los que cualquiera aplicación o servicio lanzado en los Estados Unidos puede hacer uso. Esto podría ser visto como una

fortaleza de las aplicaciones o servicios lanzados en Estados Unidos, o podría ser visto como una oportunidad puesto que dichos servicios permiten hacer cosas que sin ellos no sería posible hacer. Sin embargo, dado que cualquier aplicación o servicio lanzado en Estados Unidos puede hacer uso de dichos servicios, nadie lo considera una fortaleza o una oportunidad propia, sino más bien una fortaleza o una oportunidad del territorio. Además, si la aplicación no se lanza solamente en Estados Unidos sino que se lanza también en otros países, en esos otros países no se podría hacer uso de dichos servicios exclusivamente americanos, de modo que en el caso de estar utilizando alguno nos veríamos obligados a recortar funcionalidad al aterrizar o lanzar en otro territorio.

Decíamos que Estados Unidos es el mercado startups más competitivo del mundo. Esto lo convierte en el territorio con mayor número de competidores, y con los competidores más duros, es decir, en el territorio con más y más duras amenazas. Además, nuestro desconocimiento del sistema americano a la hora de buscar financiación o de promocionar un producto, y el hecho de no contar con ningún contacto allí, son serias debilidades para con el mercado estadounidense.

En conclusión: dado que el servicio es igual de adecuado para el mercado español que para el mercado estadounidense, no tiene sentido emprender nada en Estados Unidos si ni siquiera se puede destacar en España.

En cuanto a la contextualización temporal, está claro estamos en la época de claro apogeo de las aplicaciones móviles, y la nuestra es una aplicación móvil. De modo que esta circunstancia sería vista como una oportunidad. El hecho de que la aproximación de quienes cuestionan el modelo tradicional de contratación de servicios en línea sea tímida y sin llegar nunca a nada, da idea de que la llegada de la nueva generación de servicios de este tipo está al caer, y de

que será clave (oportunidad) posicionarse dentro de esa nueva generación de servicios, que seguro será una competencia temible para los servicios tradicionales, desde un principio. Cuando el mercado ya ha sido repartido, es mucho más difícil: la clave es ser uno de los que se lo reparten.

2.2. Alcance del proyecto

El alcance del proyecto incluye las siguientes funcionalidades:

Gestión por parte de los profesionales de su propio perfil: esto les permitirá actualizar sus datos personales, sus datos de contacto o sus datos de ubicación, cambios que se verán reflejados inmediatamente en el geodirectorio de profesionales.

Búsqueda de profesionales cercanos por parte de los clientes potenciales: esto les permitirá visualizar en un mapa los profesionales cercanos, acceder a su perfil o entablar contacto con ellos.

Gestión y visualización de chats: esto permitirá a todos los usuarios administrar sus propios chats con terceros.

Intercambio de mensajes del cliente potencial con profesionales: esto le permitirá al cliente principal, por ejemplo, establecer contacto al iniciar una conversación (con el envío del primer mensaje).

Intercambio de mensajes del profesional con clientes potenciales: esto le permitirá al profesional ser contactado por potenciales clientes interesados en sus servicios. Para el

profesional es tan sencillo como tener ejecutando la aplicación de fondo y esperar a ser notificado de mensajes de clientes potenciales nuevos o recurrentes.

Trazar el recorrido completo de los mensajes desde que se reciben en el servidor hasta que se entregan al dispositivo de destino, o incluso hasta que son leídos: esto permitirá la total fiabilidad de nuestro sistema de mensajería instantánea.

Dejar preparadas las bases para incluir en el sistema de mensajería gestión de la presencia de los usuarios, o incluso de eventos en tiempo real (en línea, escribiendo, etc.), todo con el fin de proporcionar a los usuarios un sistema de mensajería instantánea de última generación.

Creación y administración de tareas: esto permitirá luego asociar un pago o una valoración a la realización de la tarea, y también listar las tareas y su estado (creada, pagada, valorada).

Pago de tareas: esto permitirá al cliente pagar al profesional el precio de la tarea a través de PayPal, nuestro proveedor de pagos.

Valoración de tareas: esto permitirá al cliente valorar el desempeño del profesional en la tarea, una vez se haya autorizado el pago del precio de esta. De esto modo lo que se pretende lograr es que todas las valoraciones de tareas correspondan a tareas realizadas efectivamente y por ende pagadas.

Obtener traza de cada tarea: de esta forma les será sencillo tanto a clientes o consumidores como a profesionales ubicar en el tiempo los diferentes eventos asociados a la tarea en cuestión (fecha y hora de creación, fecha y hora de autorización del pago, fecha y hora de publicación de la valoración).

Listar tareas para el cliente o consumidor: las tareas que se muestran son siempre las que él mismo solicitó, las cuales se pueden mostrar todas o solamente las correspondientes a las realizadas por determinado profesional.

Listar tareas para el profesional: las tareas que se muestran son siempre las que él mismo realizó, las cuales se pueden mostrar todas o solamente las solicitadas por determinado consumidor.

Registro de usuario como consumidor o profesional: esto definirá de forma definitiva su rol dentro del micromercado virtual.

Garantizar el cumplimiento de las políticas de acceso de los diferentes tipos de usuario (consumidor, profesional, anónimo).

Invitaciones: esto permitirá determinar quiénes son aquellos usuarios cuyas invitaciones se han traducido en registros de nuevos usuarios, para después recompensarlos en función de la vigente campaña de marketing.

Garantizar el cumplimiento de las políticas de pago: sin duda, uno de los requerimientos más importantes del sistema.

2.3. Objetivos

Los objetivos se pueden clasificar en tres grupos: los objetivos del proyecto, los objetivos de la aplicación, y mis objetivos personales.

En primer lugar, los objetivos del proyecto son tres: que la aplicación cumpla sus propios objetivos, que yo cumpla mis propios objetivos y que la memoria documente correctamente el

trabajo realizado. En particular, la memoria deberá cubrir todos los apartados que aparecen en el índice, y deberá desarrollar con especial profundidad y rigor los apartados de especificación, diseño, implementación y pruebas, puesto que son los más importantes.

En segundo lugar, los objetivos de la aplicación son dos: por un lado, permitir al cliente encontrar al profesional adecuado para que le realice una tarea, permitir a cliente y profesional comunicarse mediante mensajería instantánea, y permitir al cliente pagar al profesional; por otro lado, la utilidad y el valor, que la aplicación sea útil y valiosa para sus usuarios finales. El objetivo de la aplicación no es, sin embargo, tener un sistema listo para ser lanzado al público, sino más bien la elaboración de un prototipo que permita demostrar funcionalidad.

En tercer lugar, mi objetivo personal principal es aprovechar esta oportunidad que me brinda la universidad para formarme y aprender lo máximo posible: aprender nuevas metodologías de desarrollo software, nuevas filosofías o lenguajes de patrón para el diseño y la implementación de software, y nuevas aproximaciones a la hora de probar un sistema; aprender también todas las tecnologías necesarias para la implementación de este proyecto que, o bien no conozco, o bien no he utilizado antes. Aprender sobre todo de los libros, pero también de la experiencia. Solo así podré cumplir mi otro gran objetivo personal, que es el de conseguir implementar un sistema que considero complejo y ambicioso.

3. Metodología

La metodología de desarrollo software que se aplicará en este proyecto será ágil y, por tanto, basada en el desarrollo iterativo e incremental. En particular, se combinarán dos metodologías de desarrollo ágil de software, una de segunda generación que es desarrollo guiado por el comportamiento o Behavior-Driven Development (BDD), y otra de primera generación que es Scrum.

3.1. Desarrollo guiado por el comportamiento

En primer lugar, se aplicará desarrollo guiado por el comportamiento hasta llegar a la especificación de las pruebas de aceptación. El proceso se desarrollará de la siguiente manera:

1. En primer lugar, se identificarán los objetivos del negocio para con la aplicación, que muchas veces serán diferentes formas de generar ingresos, o de generar más ingresos, directa o indirectamente.

2. En segundo lugar, se buscarán las funcionalidades que van a permitir lograr los objetivos del negocio. Una vez encontradas se definirán formalmente, y las muy grandes se dividirán en historias de usuario, las cuales serán definidas formalmente exactamente igual que el resto de funcionalidades.

3. En tercer lugar se ilustrarán las funcionalidades con ejemplos, los cuales, una vez formalizados, se convertirán en especificaciones de las pruebas de aceptación.

De lo obtenido al realizar este proceso, lo que realmente será valioso serán la especificación formal de las funcionalidades, y la especificación formal de las pruebas de aceptación. Por un lado, las funcionalidades obtenidas y, más en particular, su especificación formal, serán utilizadas directamente por Scrum, como se verá en la siguiente sección de este apartado. Por otro lado, las especificaciones de las pruebas de aceptación guiarán el diseño y la implementación de las pruebas, proceso que se realizará tras el diseño y la implementación del código. Es decir, que primero se diseñará e implementará el código, y después se diseñarán e implementarán las pruebas.

3.2. Scrum

En segundo lugar, se aplicará Scrum para priorizar las funcionalidades obtenidas al aplicar desarrollo guiado por el comportamiento y para planificar los sprints. El proceso se desarrollará de la siguiente manera:

1. En primer lugar, a las funcionalidades obtenidas en el apartado anterior se les asignará una prioridad, y se añadirán al Product Backlog como Product Backlog Items o PBI.

2. En segundo lugar, se planificarán los sprints a partir de los PBIs del Product Backlog. Dichos sprints tendrán una duración de una semana. Al finalizar cada sprint se revisará el trabajo realizado y el no realizado, es decir, los PBIs completados y los no completados, y se probará el funcionamiento del trabajo completado.

Nótese que dado que el Product Owner, el ScrumMaster y el equipo de desarrollo son la misma persona, en este proyecto se hará una aplicación parcial de Scrum. Esta aplicación parcial de Scrum afectará principalmente a las reuniones, que no se celebrarán: ni la de Daily Scrum; ni la

de Scrum de Scrum, que es la que menos sentido tiene; ni las de planificación, revisión o retrospectiva del Sprint. No obstante, se realizará la planificación y la revisión de todos los sprints, pero sin reunión de por medio.

3.3. Una vez dentro de los sprints: UML y diseño guiado por el dominio

Una vez dentro de un sprint, debemos realizar estas cuatro tareas por estricto orden: terminar de especificar las funcionalidades, diseñarlas, implementarlas y probarlas.

En primer lugar, se terminarán de especificar las funcionalidades. Al aplicar desarrollo guiado por el comportamiento lo que hicimos fue obtener una especificación de todas las funcionalidades, pero no de los detalles de cada funcionalidad particular. Ahora especificaremos los detalles de cada funcionalidad particular utilizando UML.

En segundo lugar, se diseñarán las funcionalidades. Para ello se aplicará diseño guiado por el dominio o Domain-Driven Design (DDD) a partir de las especificaciones completas de las funcionalidades obtenidas en el apartado anterior. Parte de los resultados obtenidos aplicando diseño guiado por el dominio se representarán en UML.

En tercer lugar, se implementarán las funcionalidades.

Por último, se probarán las funcionalidades. Para ello se implementarán pruebas a partir de la especificación de las pruebas de aceptación. Recordemos que dicha especificación la obtuvimos al aplicar desarrollo guiado por el comportamiento.

4. Planificación, costes y riesgos

Este apartado se dividirá en tres secciones: planificación, análisis de costes y análisis de riesgos.

4.1. Planificación

Esta sección se divide en dos: por un lado, la planificación inicial; por otro lado, la situación final.

4.1.1. Planificación inicial

Antes de todo, debo realizar un breve comentario aclaratorio. En julio de 2015 matriculé un proyecto final de carrera que nada tenía que ver con este. En marzo de 2016 decidí cambiar ese proyecto final de carrera por este otro, de modo que esta es la verdadera fecha de inicio del proyecto, y no la fecha en que lo matriculé.

Ahora sí, vamos con la planificación inicial:

- Desde el 1 de marzo al 28 de abril de 2016: formación, a razón de un libro por semana, mientras trabajaba en otro proyecto. Varios de estos libros se pueden encontrar en la bibliografía. Añado también que la formación ha sido una constante durante todo el proyecto, a razón de unas 40 horas semanales, pero no la incluyo ni en el resto de la planificación ni en el análisis de costes.
- Semana 1 (del 28 al 3 de abril): definición y planificación.
- Semana 2 (del 4 al 10 de abril): diseño de la interfaz de usuario.

- Semana 3 (del 11 al 17 de abril): especificación completa del proyecto aplicando desarrollo guiado por el comportamiento; creación de los Product Backlog Items del Product Backlog a partir de las especificaciones de funcionalidades e historias de usuario.
- Semana 4 (del 18 al 25 de abril): sprint 1: Mensajería instantánea, servidor (parte 1).
- Semana 5 (del 26 de abril al 1 de mayo): sprint 2: Mensajería instantánea, servidor (parte 2).
- Semana 6 (del 2 al 6 de mayo): sprint 3: Mensajería instantánea, servidor (parte 3).
- Semana 7 (del 7 al 13 de mayo): sprint 4: Mensajería instantánea, servidor (parte 4).
- Semana 8 (del 14 al 20 de mayo): sprint 5: Mensajería instantánea, cliente (parte 1).
- Semana 9 (del 21 al 27 de mayo): sprint 6: Mensajería instantánea, cliente (parte 2).
- Semana 10 (del 28 de mayo al 3 de junio): sprint 7: Pago de tareas, servidor (parte 1).
- Semana 11 (del 4 de junio al 10 de junio): sprint 8: Pago de tareas, servidor (parte 2).
- Semana 12 (del 11 al 18 de junio): sprint 9: Pago de tareas, cliente.
- Semana 13 (del 19 al 25 de junio): sprint 10: Geodirectorio de profesionales, servidor (parte 1).
- Semana 14 (del 26 de junio al 2 de julio): sprint 11: Geodirectorio de profesionales, servidor (parte 2).
- Semana 15 (del 3 al 10 de julio): sprint 12: Geodirectorio de profesionales, cliente (parte 1).
- Semana 16 (del 11 al 17 de julio): sprint 13: Geodirectorio de profesionales, cliente (parte 2).
- Semana 17 (del 18 al 24 de julio): sprint 14: Identidad y acceso, servidor.
- Semana 18 (del 25 al 31 de julio): sprint 15: Identidad y acceso, cliente.

- Semana 19 (del 1 al 7 de julio): sprint 16: Invitaciones, servidor (parte 1).
- Semana 20 (del 8 al 14 de agosto): sprint 17: Invitaciones, servidor (parte 2).
- Semana 21 (del 15 al 21 de agosto): sprint 18: Valoraciones de tareas, servidor.
- Del 22 de agosto al 22 de septiembre: Memoria.

4.1.2. Situación final

Lo que ha ocurrido realmente ha sido lo siguiente:

- Desde el 1 de marzo al 28 de abril de 2016: formación, a razón de un libro por semana, mientras trabajaba en otro proyecto. Varios de estos libros se pueden encontrar en la bibliografía. Añado también que la formación ha sido una constante durante todo el proyecto, a razón de unas 40 horas semanales, pero no la incluyo ni en el resto de la planificación ni en el análisis de costes.
- Semana 1 (del 28 al 3 de abril): definición y planificación.
- Semana 2 (del 4 al 10 de abril): diseño de la interfaz de usuario.
- Semana 3 (del 11 al 17 de abril): especificación completa del proyecto aplicando desarrollo guiado por el proyecto; creación de los Product Backlog Items del Product Backlog a partir de las especificaciones de funcionalidades y tareas de usuario.
- Semana 4 (del 18 al 25 de abril): sprint 1: Mensajería instantánea, servidor (parte 1).
- Semana 5 (del 26 de abril al 1 de mayo): sprint 2: Mensajería instantánea, servidor (parte 2).
- Semana 6 (del 2 al 6 de mayo): sprint 3: Mensajería instantánea, servidor (parte 3).
- Semana 7 (del 7 al 13 de mayo): sprint 4: Mensajería instantánea, servidor (parte 4).

- Semana 8 (del 14 al 20 de mayo): sprint 5: Mensajería instantánea, servidor (parte 5).
- Semana 9 (del 21 al 27 de mayo): sprint 6: Mensajería instantánea, cliente (parte 1).
- Semana 10 (del 28 de mayo al 3 de junio): sprint 7: Mensajería instantánea, cliente (parte 2) .
- Semana 11 (del 4 de junio al 10 de junio): sprint 8: Mensajería instantánea, cliente (parte 3).
- Semana 12 (del 11 al 18 de junio): sprint 9: Pago de tareas, servidor (parte 1).
- Semana 13 (del 19 al 25 de junio): sprint 10: Pago de tareas, servidor (parte 2).
- Semana 14 (del 26 de junio al 2 de julio): sprint 11: Pago de tareas, cliente.
- Semana 15 (del 3 al 10 de julio): sprint 12: Geodirectorio de profesionales, servidor (parte 1).
- Semana 16 (del 11 al 17 de julio): sprint 13: Geodirectorio de profesionales, servidor (parte 2).
- Semana 17 (del 18 al 24 de julio): sprint 14: Geodirectorio de profesionales, cliente (parte 1).
- Semana 18 (del 25 al 31 de julio): sprint 15: Geodirectorio de profesionales, cliente (parte 2).
- Semana 19 (del 1 al 7 de agosto): sprint 16: Identidad y acceso, servidor.
- Semana 20 (del 8 al 14 de agosto): sprint 17: Identidad y acceso, cliente.
- Semana 21 (del 15 al 21 de agosto): sprint 18: Invitaciones, servidor (parte 1).
- Semana 22 (del 22 al 28 de agosto): sprint 19: Invitaciones, servidor (parte 2).
- Semana 23 (del 29 al 4 de septiembre): sprint 20: Valoraciones de tareas, servidor (parte 1).
- Del 5 de septiembre al 22 de septiembre: Memoria.

La diferencia radica en que ha habido un retraso de dos semanas en la parte de Mensajería instantánea, una semana imputable al servidor y otra semana imputable al cliente.

La situación final es que se han implementado completamente los componentes de Mensajería instantánea, Pago de tareas, Geodirectorio de profesionales y Autenticación y acceso, y la parte del servidor de Invitaciones y de Valoraciones de tareas: lo mismo que en la planificación inicial.

4.2. Análisis de costes

Para la realización del análisis de costes, básicamente se han tenido en cuenta cuatro grupos de costes: los costes directos de personal, los costes indirectos del proyecto, los costes del software y los costes del hardware.

En primer lugar, los costes de personal incluirían los sueldos y los salarios de los integrantes del equipo de desarrollo. En este caso, el único integrante del equipo de desarrollo soy yo. Sin embargo, he desempeñado diferentes roles a lo largo de la realización de este proyecto software. En particular, he desempeñado el rol de propietario del producto, de arquitecto software, de desarrollador web, de desarrollador móvil y de diseñador de interfaces de usuario. El número de horas que he dedicado en cada rol, lo obtenemos de la planificación:

- Como propietario del producto: 1 semana (la semana 1: Definición y planificación).
- Como diseñador gráfico: 1 semana (la semana 2: Diseño de la interfaz de usuario).
- Como arquitecto software: 7 semanas (la mitad de 14 semanas, que son las dedicadas a la especificación completa de la aplicación y a los sprints que conciernen al servidor).
- Como desarrollador web: 7 semanas (que es la otra mitad de las 14 semanas).

- Como cliente móvil: 7 semanas (que son las dedicadas a los sprints que conciernen al cliente).

El detalle de los costes de personal sería el siguiente:

Rol del trabajador	**Horas dedicadas**	**Coste/hora**	**Coste**
Propietario del producto	1 semana * 5 días/semana * 8 horas/día = 40 horas	20€	800€
Arquitecto software	7 semanas * 5 días/semana * 8 horas/día = 280 horas	18€	5040€
Desarrollador web	7 semanas * 5 días/semana * 8 horas/día = 280 horas	12€	3360€
Desarrollador móvil	7 semanas * 5 días/semana* 8 horas/día = 280 horas	15€	4200€
Diseñador gráfico	1 semana * 5 días/semana* 8 horas/día = 40 horas	10€	400€
Total	920 horas		13800€

Las horas de formación no se incluyen en ningún caso.

En segundo lugar, los costes indirectos del proyecto serían costes tales como la electricidad, el agua, el gas, el acceso a Internet, o el alquiler de la oficina. Dado que este proyecto se ha realizado en la casa de su autor, considero que contabilizar estos gastos no tiene sentido.

En tercer lugar, no hay costes de software, puesto que todo el software utilizado es gratuito.

En cuarto lugar, los costes del hardware se reducen a los costes de los siete droplets estándar de Digital Ocean: uno, para el geodirectorio de profesionales, con un coste de 80$ al mes; otros seis, para los otros seis componentes, a razón de 10$ al mes cada uno. Dichos droplets se han utilizado para publicar los diferentes servicios de la aplicación en Internet, para que así puedan ser accedidos por el cliente móvil. Teniendo en cuenta que estos droplets no se han contratado hasta un mes antes de la entrega, y realizando la conversión estimando que el valor de un dólar equivale a un euro, la tabla de costes nos queda como se detalla a continuación:

Elemento hardware	**Cantidad**	**Meses**	**Coste/mes**	**Coste**
Droplet estándar tipo I	1	1 mes	80€	80€
Droplet estándar tipo II	6	1 mes	10€	60€
Total	7	1 mes		140€

Aclaro que no incluyo ni los costes del portátil ni los costes del teléfono móvil que he utilizado durante el desarrollo, puesto que son los míos de uso personal y considero que no hay que imputar sus costes a los del proyecto.

Al agregar los costes directos de personal y los costes del hardware, el coste total del proyecto asciende a **13940€**.

4.3. Análisis de riesgos

Esta sección se dividirá en dos partes. En la primera nombraré, clasificaré y mediré todos los riesgos identificados en este proyecto software. En la segunda, presentaré los planes de mitigación y de contención en caso de que estos riesgos llegaran a materializarse.

4.3.1. Identificación, clasificación y medición de los riesgos

Para la elaboración de esta sección me he basado en el artículo *Lista de comprobación de riesgos software* elaborada por Carlos Blanco para la Universidad de Cantabria [Blanco].

A continuación, el detalle de todos los riesgos identificados.

4.3.1.1. Planificación optimista, “mejor caso”

Tipo de riesgo: Elaboración de la planificación

Impacto (0..10): 10

Probabilidad (0..100%): 25

Magnitud: 250

En el contexto de este proyecto: no he considerado la posibilidad de quedarme encallado en algún punto, y eso, tratándose de tecnologías en las que soy inexperto, es una asunción arriesgada.

4.3.1.2. No se puede construir un producto de tal envergadura en el tiempo asignado

Tipo de riesgo: Elaboración de la planificación

Impacto (0..10): 10

Probabilidad (0..100%): 25

Magnitud: 250

En el contexto de este proyecto: existe la posibilidad que dada mi inexperiencia, y mi elección de un sistema complejo para aprender con él, yo como única persona implicada en este proyecto no sea capaz de construir semejante proyecto en el tiempo asignado.

4.3.1.3. El esfuerzo es mayor que el estimado

Tipo de riesgo: Elaboración de la planificación

Impacto (0..10): 8

Probabilidad (0..100%): 30

Magnitud: 240

En el contexto de este proyecto: desconozco el esfuerzo real necesario para implementar funcionalidades que nunca he implementado antes con tecnologías que nunca he utilizado antes. Por tanto, para estimar, he intentado comparar con lo que sí conozco y se parece mínimamente, lo que ocurre es que cuando se hace esto se suele infravalorar el esfuerzo de lidiar con lo desconocido.

4.3.1.4. Las áreas desconocidas del producto llevan más tiempo del esperado en el diseño y en la implementación

Tipo de riesgo: Elaboración de la planificación

Impacto (0..10): 6

Probabilidad (0..100%): 35

Magnitud: 240

En el contexto de este proyecto: relacionado con puntos anteriores, quedarse encallado en algún punto es probable, así como infravalorar el esfuerzo necesario para dominar una tecnología que nos es nueva.

4.3.1.5. La planificación es demasiado mala para ajustarse a la velocidad de desarrollo deseada

Tipo de riesgo: Organización y gestión

Impacto (0..10): 8

Probabilidad (0..100%): 25

Magnitud: 200

En el contexto de este proyecto: por mala en este caso quiero decir optimista, tan optimista que no sea posible seguirla.

4.3.1.6. La curva de aprendizaje para la nueva herramienta de desarrollo es más larga de lo esperado

Tipo de riesgo: Ambiente/infraestructura de desarrollo

Impacto (0..10): 7

Probabilidad (0..100%): 35

Magnitud: 245

Contexto de este proyecto: en este proyecto hay casi una decena de tecnologías que nunca he utilizado antes, y esta posibilidad de que en alguna de ellas la curva de aprendizaje sea más larga de lo esperado es real.

4.3.1.7. No se ha solicitado información al usuario, por lo que el producto al final no se ajusta a las necesidades del usuario, y hay que volver a crear el producto

Tipo de riesgo: Usuarios finales

Impacto (0..10): 10

Probabilidad (0..100%): 1

Magnitud: 10

En el contexto de este proyecto: en efecto, estamos convencidos de que este producto responde a una necesidad real, pero nunca se puede estar seguro al ciento por ciento.

4.3.1.8. El desarrollo de una interfaz de usuario inadecuada requiere volver a diseñarla y a implementarla

Tipo de riesgo: Producto

Impacto (0..10): 4

Probabilidad (0..100%): 5

Magnitud: 60

En el contexto de este proyecto: en efecto, estamos convencidos de que la interfaz de usuario de esta aplicación es adecuado, pero nunca se puede estar seguro al ciento por ciento.

4.3.1.9. Los requisitos para crear interfaces con otros sistemas, otros sistemas complejos, u otros sistemas que no están bajo el control del equipo de desarrollo suponen un diseño, implementación y prueba no previstos

Tipo de riesgo: Producto

Impacto (0..10): 6

Probabilidad (0..100%): 20

Magnitud: 120

En el contexto de este proyecto: este es un riesgo muy real debido a las elevadas complejidades de integración entre sistemas de nuestro proyecto, tanto con sistemas externos, como de los sistemas propios entre ellos.

4.3.1.10. Las tareas preliminares (por ejemplo, formación, finalización de otros proyectos, adquisición de licencias) no se han completado a tiempo

Tipo de riesgo: Personal

Impacto (0..10): 9

Probabilidad (0..100%): 50

Magnitud: 450

En el contexto de este proyecto: en nuestro caso, existe una labor muy importante de formación por mi parte que es posible que no pueda completar a tiempo.

4.3.1.11. La falta de la especialización necesaria aumenta los defectos y la necesidad de repetir el trabajo

Tipo de riesgo: Personal

Impacto (0..10): 4

Probabilidad (0..100%): 15

Magnitud: 60

En el contexto de este proyecto: mi falta de especialización en más de una decena de tecnologías estoy convencido de que aumentará los defectos o incluso que me obligará a repetir parte del trabajo.

4.3.1.12. El personal necesita un tiempo extra para acostumbrarse a trabajar con herramientas o entornos nuevos

Tipo de riesgo: Personal

Impacto (0..10): 5

Probabilidad (0..100%): 20

Magnitud: 100

En el contexto de este proyecto: sin duda es muy probable que necesite tiempo extra para acostumbrarme a trabajar con herramientas y entornos nuevos.

4.3.1.13. No hay suficiente personal disponible para el proyecto

Tipo de riesgo: Personal

Impacto (0..10): 8

Probabilidad (0..100%): 30

Magnitud: 240

En el contexto de este proyecto: soy el único integrante del equipo de desarrollo, y además carezco de experiencia en muchos de sus múltiples frentes. Sin duda, otro miembro en el equipo con experiencia en este tipo de sistemas se traduciría en unas probabilidades mucho más elevadas de éxito.

4.3.1.14. El personal trabaja más lento de lo esperado

Tipo de riesgo: Personal

Impacto (0..10): 8

Probabilidad (0..100%): 25

Magnitud: 200

En el contexto de este proyecto: sin duda, la inexperiencia y la falta de práctica hacen que en muchas ocasiones uno trabaje más lento de lo esperado.

4.3.1.15. Un diseño demasiado complejo exige tener en cuenta complicaciones innecesarias e improductivas en la implementación

Tipo de riesgo: Diseño e implementación

Impacto (0..10): 7

Probabilidad (0..100%): 35

Magnitud: 245

En el contexto de este proyecto: todo apunta a que la elección de un sistema tan complejo va a generar complicaciones innecesarias e improductivas en la implementación.

4.3.1.16. La utilización de metodologías desconocidas deriva en un periodo extra de formación y tener que volver atrás para corregir los errores iniciales cometidos en la metodología

Tipo de riesgo: Diseño e implementación

Impacto (0..10): 8

Probabilidad (0..100%): 30

Magnitud: 240

En el contexto de este proyecto: no solamente las tecnologías desconocidas, sino también las metodologías desconocidas puede que requieran de un tiempo extra de formación con respecto al inicialmente previsto, o incluso puede que sea necesario volver atrás en algún momento para corregir los errores inicialmente cometidos al aplicar la metodología.

4.3.1.17. Los componentes desarrollados por separado no se pueden integrar de forma sencilla, teniendo que volver a diseñar y repetir algunos trabajos

Tipo de riesgo: Diseño e implementación

Impacto (0..10): 8

Probabilidad (0..100%): 20

Magnitud: 160

En el contexto de este proyecto: la elección de crear un sistema con múltiples componentes dispara la complejidad. Por ejemplo, es posible que los componentes integrados por separado no se puedan integrar de forma sencilla.

4.3.1.18. La gestión de riesgos del proyecto software consume más tiempo del esperado

Tipo de riesgo: Diseño e implementación

Impacto (0..10): 10

Probabilidad (0..100%): 20

Magnitud: 1

En el contexto de este proyecto: es probable que, dado el alto riesgo de este proyecto, se tenga que dedicar tiempo adicional a la gestión de riesgos cuando estos se materializan.

4.3.2. Planes de medición y de contingencia

Como vemos, es este un proyecto de elevado riesgo. Las razones son dos: por un lado, la ambición del cliente en cuanto al número de requisitos funcionales de la aplicación; por otro lado, la ambición del desarrollador en cuanto a la complejidad del sistema a construir, a la que hay que añadir su inexperiencia en la implementación de este tipo de sistemas. Debo aclarar que el cliente y el desarrollador son la misma persona, es decir, quien escribe estas líneas.

Los planes de mitigación y de contención para estos riesgos son como sigue:

Por un lado, para lo riesgos causados por el elevado número de requisitos funcionales, el plan de mitigación consiste en la reducción del número de casos de uso implementados para el mismo requisito funcional; en cuanto al plan de contención, este consiste en recortar funcionalidades de la aplicación para esta entrega, puesto que la fecha de entrega no puede posponerse.

Por otro lado, para los riesgos causados por la elevada complejidad del sistema y la inexperiencia del desarrollador, el plan de mitigación consiste en disminuir el número de componentes a integrar, fusionando algunos de ellos, y la dedicación de horas extra no remuneradas de formación por parte del desarrollador; en cuanto al plan de contención, este consiste en la dedicación de horas extra por parte del desarrollador no solamente para formarse sino también para programar, y la contratación de algún experto por lo menos hasta la fecha de entrega de la aplicación.

5. Análisis: especificación

Este apartado está dividido en cuatro secciones: la primera, destinada a introducir el desarrollo guiado por el comportamiento; la segunda, en la que se especifican los requerimientos funcionales o funcionalidades del sistema; la tercera, en la que se especifican los requerimientos no funcionales; y la cuarta, que contiene los diagramas de casos de uso.

5.1. Introducción al desarrollo guiado por el comportamiento

Para la realización de las secciones que especificarán los requerimientos funcionales y no funcionales del sistema, nos basaremos en el análisis de requerimientos que realizan quienes practican desarrollo guiado por el comportamiento o Behavior-Driven Development (BDD). Ellos se refieren a los requerimientos como features, pero a todos los efectos son sinónimos.

Sobre el significado del término feature en español

Feature suele significar característica, pero cuando hablamos de un sistema también significa funcionalidad. Las características que debe poseer un sistema las definen los requerimientos no funcionales, mientras que las funcionalidades que debe poseer un sistema las definen los requerimientos funcionales. Dado que feature significa tanto requerimiento no funcional como requerimiento funcional, lo trataremos como sinónimo de requerimiento.

El formato en el que expresan los requerimientos quienes practican BDD es el siguiente:

Para <realizar un objetivo del negocio>

Como <parte interesada>

Quiero <algo>

En efecto, la finalidad expresa un objetivo de negocio, y se pone primero porque se considera lo más importante. Esto resulta decisivo luego a la hora de asignar prioridades.

Dicho esto, vamos con los requerimientos.

5.2. Requerimientos funcionales

En esta sección, para cada requerimiento funcional o funcionalidad obtenido aplicando desarrollo guiado por el comportamiento, detallo la siguiente información:

1. En primer lugar, la especificación formal del requerimiento siguiendo el formato propuesto por BDD.
2. En segundo lugar, la prioridad asignada al requerimiento funcional antes de introducirlo como Product Backlog Item en el Product Backlog de Scrum.
3. Por último, desvelo si la funcionalidad ha sido o no finalmente implementada.

5.2.1. Permitir a los usuarios registrarse

Especificación formal:

> Para poder empezar a usar el servicio y así contratar profesionales
>
> Como usuario
>
> Quiero que el sistema me permita registrarme.

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.2. Permitir a los usuarios autenticarse

Especificación formal:

Para poder hacer uso de todas aquellas funcionalidades que requieren de autenticación

Como usuario

Quiero que el sistema me permita autenticarme.

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.3. Recordar credenciales de los usuarios

Especificación formal:

Para evitar tener que introducir los credenciales una y otra vez

Como usuario

Quiero que el sistema recuerde mis credenciales

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.4. No permitir el uso del cliente móvil a los usuarios si no están autenticados

Especificación formal:

Para evitar que los usuarios puedan utilizar la aplicación sin autenticarse

Como propietario

Quiero que la aplicación móvil exija la autenticación del usuario para ser usada

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.5. Permitir a los usuarios recuperar su contraseña

Especificación formal:

Para poder seguir utilizando el servicio aunque olvide mi contraseña

Como usuario

Quiero que el sistema me permita recuperar mi contraseña

Prioridad asignada: 2

Efectivamente implementado: No

5.2.6. Verificación de la cuenta de email facilitada por los usuarios al registrarse

Especificación formal:

Para evitar una base usuarios plagada de molestos multinick que no están por la labor

Como usuario, propietario

Quiero que el sistema verifique las cuentas de email facilitadas por los usuarios al registrarse

Prioridad asignada: 2

Efectivamente implementado: No

5.2.7. Permitir a los clientes obtener un informe con los datos personales de un profesional

Especificación formal:

Para conocer mejor a un profesional

Como cliente

Quiero que el sistema me permita ver sus datos personales

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.8. Permitir a los profesionales obtener un informe con los datos personales de un cliente

Especificación formal:

Para conocer mejor a un cliente

Como profesional

Quiero que el sistema me permita ver sus datos personales

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.9. Permitir a los usuarios facilitar datos de contacto de conocidos a los que desean que el sistema envíe una invitación de su parte

Especificación formal:

Para poder invitar

Como usuario

Quiero que el sistema me permita facilitar datos de contacto de conocidos a los que deseo que el sistema envíe una invitación de mi parte

Prioridad asignada: 4

Efectivamente implementado: No

5.2.10. Envío de invitaciones a conocidos de usuarios

Especificación formal:

> **Para** poder beneficiarme del sistema de recompensa a invitadores
>
> Como usuario
>
> Quiero que el sistema envíe invitaciones a los conocidos que le indique
>
> **Para** incrementar la base de usuarios de la aplicación
>
> Como propietario
>
> Quiero que el sistema envíe invitaciones a aquellas personas que nos indiquen los usuarios

Prioridad asignada: 4

Efectivamente implementado: No

5.2.11. Detectar y registrar cuando un usuario se ha registrado a través de la invitación de otro

Especificación formal:

> Para que el sistema conozca los frutos de mi esfuerzo a la hora de invitar a la gente a que pruebe la aplicación, y con esperanzas de que me recompense por ello
>
> Como usuario

Quiero que el sistema detecte y registre cuando un usuario se registra a través de una de mis invitaciones

Prioridad asignada: 4

Efectivamente implementado: No

5.2.12. Recompensar a aquellos usuarios cuyas invitaciones han sido utilizadas por otros usuarios para registrarse. La recompensa consistirá en una participación en la tasa que cobre el sistema a los pagos realizados o percibidos por el usuario invitado

Especificación formal:

Para que el sistema premie mi esfuerzo y mi éxito a la hora de invitar a personas a que prueben la aplicación

Como usuario

Quiero que el sistema me recompensa toda vez que un usuario utiliza una de mis invitaciones para registrarse

Para proporcionar a los usuarios una gran motivación para que promuevan la aplicación allá donde vayan

Como propietario

Quiero que el sistema recompense a aquellos usuarios cuyas invitaciones han sido utilizadas por otros usuarios para registrarse

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.13. Permitir a los usuarios obtener un informe con el total percibido por invitaciones exitosas

Especificación formal:

Para poder estar al corriente de los frutos de mi actividad invitadora

Como usuario

Quiero que el sistema me permita obtener un informe con el total percibido por invitaciones exitosas.

Prioridad asignada: 3

Efectivamente implementado: No

5.2.14. Permitir al cliente iniciar un chat con un profesional

Especificación formal:

Para poder solicitar una tarea

Como cliente

Quiero que el sistema me permita iniciar un chat con un profesional

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.15. Permitir al cliente el envío de mensajes a un profesional

Especificación formal:

Para poder explicarle a un profesional lo que necesito

Como cliente

Quiero que el sistema me permita enviarle mensajes

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.16. Permitir al profesional el envío de mensajes a un cliente

Especificación formal:

Para poder explicarle a un cliente lo que ofrezco

Como profesional

Quiero que el sistema me permita enviarle mensajes

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.17. Envío de mensajes a un cliente que le han sido enviados por un profesional

Especificación formal:

Para poder saber qué me ofrece un profesional

Como cliente

Quiero que el sistema me haga llegar sus mensajes

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.18. Envío de mensajes a un profesional que le han sido enviados por un cliente

Especificación formal:

Para poder saber qué necesita un cliente

Como profesional

Quiero que el sistema me haga llegar sus mensajes

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.19. Notificación al usuario de que el mensaje que ha enviado ha sido recibido por el servidor

Especificación formal:

> Para confirmar que no tengo un problema de conexión con mi teléfono que me está dificultando las conversaciones con otros usuarios
>
> Como usuario
>
> Quiero ser notificado cuando mis mensajes lleguen al servidor.

Prioridad asignada: 1

Efectivamente implementado: No (aunque el evento se registra en el servidor)

5.2.20. Notificación al usuario de que el mensaje que ha enviado ha sido entregado al dispositivo de su interlocutor

Especificación formal:

Para confirmar que mi interlocutor tiene conexión en su dispositivo móvil

Como usuario

Quiero ser notificado cuando mis mensajes lleguen al dispositivo de mi interlocutor.

Prioridad asignada: 1

Efectivamente implementado: No (aunque el evento se registra en el servidor)

5.2.21. Notificación al usuario de que el mensaje que ha enviado ha sido leído por su interlocutor

Especificación formal:

Para confirmar que mi interlocutor ha visto mi mensaje

Como usuario

Quiero ser notificado cuando mi interlocutor lea mis mensajes.

Prioridad asignada: 1

Efectivamente implementado: No

5.2.22. Permitir al usuario bloquear a otros usuarios con los que ha establecido contacto

Especificación formal:

Para poder seguir usando la aplicación tranquilamente

Como usuario

Quiero poder bloquear a usuarios molestos

Prioridad asignada: 3

Efectivamente implementado: No

5.2.23. Permitir al usuario denunciar a otros usuarios con los que ha establecido contacto

Especificación formal:

Para poder expresar mi disgusto con respecto a un usuario

Como usuario

Quiero poder denunciar a usuarios desagradables

Prioridad asignada: 3

Efectivamente implementado: No

5.2.24. Desactivación de profesionales que no contestan a los mensajes que les envían los clientes para que no salgan en los informes de profesionales cercanos

Especificación formal:

Para evitar perder el tiempo con profesionales inactivos

Como cliente

Quiero que el sistema desactive a los profesionales que no contestan a los mensajes para que no salgan en los informes de profesionales cercanos

Prioridad asignada: 3

Efectivamente implementado: No

5.2.25. Permitir a los usuarios eliminar un chat

Especificación formal:

Para evitar acumular chats inactivos e irrelevantes

Como usuario

Quiero que el sistema me permita eliminar chats

Prioridad asignada: 1

Efectivamente implementado: Sí

5.2.26. Permitir al profesional especificar sus datos de ubicación

Especificación formal:

Para aparecer en el mapa de profesionales cercanos y eventualmente ser contratado

Como profesional

Quiero que el sistema me permita especificar mis datos de ubicación

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.27. Permitir al profesional actualizar sus datos de ubicación

Especificación formal:

> Para que el sistema me siga recomendando a clientes cercanos tras cambiar de ubicación
>
> Como profesional
>
> Quiero que el sistema me permita actualizar mis datos de ubicación

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.28. Permitir al profesional especificar su profesión

Especificación formal:

> Para que los usuarios puedan saber mi profesión y facilitar por ende su interés en mi labor
>
> Como profesional
>
> Quiero poder especificar mi profesión

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.29. Permitir al cliente obtener un informe con profesionales cercanos

Especificación formal:

Para poder solicitar la tarea al profesional adecuado

Como cliente

Quiero que el sistema me permita obtener un informe con profesionales cercanos

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.30. Permitir al cliente obtener un informe con profesionales cercanos de determinada profesión

Especificación formal:

> Para poder solicitar la tarea al profesional adecuado
>
> Como cliente
>
> Quiero que el sistema me permita obtener un informe con profesionales cercanos de determinada profesión

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.31. Permitir al profesional especificar la cuenta en la que desea recibir sus pagos

Especificación formal:

> Para poder recibir pagos a través de la aplicación, y así aumentar mi reputación
>
> Como profesional
>
> Quiero que el sistema me permita especificar la cuenta en la que deseo recibir mis pagos

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.32. Permitir al cliente obtener un informe con el total cobrado por un profesional a través de la aplicación

Especificación formal:

> **Para** saber cuán reputado es un profesional
>
> Como cliente
>
> Quiero que el sistema me permita obtener un informe con el total cobrado por un profesional a través de la aplicación
>
> **Para** motivar a los profesionales a que reciban los pagos a través de la aplicación

Como propietario

Quiero que sus cobros totales a través de la aplicación sean claramente visibles para los clientes

Prioridad asignada: 5

Efectivamente implementado: Solo en el servidor

5.2.33. Permitir al cliente pagar al profesional que le ha realizado una tarea

Especificación formal:

Para poder pagar a los profesionales cómodamente

Como cliente

Quiero poder pagar a través de la aplicación

Prioridad asignada: 5

Efectivamente implementado: Sí

5.2.34. Permitir a los clientes solicitar una tarea a un profesional

Especificación formal:

Para lograr que un profesional me realice una tarea

Como cliente

Quiero que el sistema me permita solicitarle la tarea a un profesional

Prioridad asignada: 3

Efectivamente implementado: No

5.2.35. Permitir a los profesionales realizar un presupuesto para una tarea solicitada por un cliente

Especificación formal:

Para ser contratado por un cliente

Como profesional

Quiero que el sistema me permita realizarle un presupuesto para las tareas solicitadas.

Prioridad asignada: 3

Efectivamente implementado: No

5.2.36. Permitir a los clientes rechazar un presupuesto realizado por un profesional

Especificación formal:

Para evitar pagar por tareas cuyo presupuesto no he aceptado

Como cliente

Quiero que el sistema me permita rechazar el presupuesto realizado por un profesional

Prioridad asignada: 3

Efectivamente implementado: No

5.2.37. Permitir a los clientes aceptar un presupuesto realizado por un profesional

Especificación formal:

Para poder pagar por tareas cuyo presupuesto he aceptado

Como cliente

Quiero que el sistema me permita aceptar el presupuesto de un profesional

Prioridad asignada: 3

Efectivamente implementado: No

5.2.38. Permitir a los profesionales indicar que ya han realizado la tarea que les encomendó el cliente

Especificación formal:

Para que el cliente compruebe el trabajo realizado y proceda al pago

Como profesional

Quiero que el sistema me permita indicar que ya he realizado la tarea que me encomendó el cliente

Prioridad asignada: 3

Efectivamente implementado: No

5.2.39. Permitir al cliente valorar el desempeño del profesional en la realización de determinada tarea

Especificación formal:

> Para poder recompensar o penalizar a un profesional en función de mi grado de satisfacción con su trabajo
>
> Como cliente
>
> Quiero que el sistema me permita valorar el desempeño de un profesional en la realización de determinada tarea

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.40. Permitir al cliente obtener un informe con las tareas que ha solicitado

Especificación formal:

Para poder controlar mi actividad en la aplicación y usarla con más tranquilidad

Como cliente

Quiero que el sistema me permita obtener un informe con las tareas que he solicitado

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.41. Permitir al profesional obtener un informe con las tareas que se le han solicitado

Especificación formal:

> Para poder controlar mi actividad en la aplicación y usarla con más tranquilidad
>
> Como profesional
>
> Quiero que el sistema me permita obtener un informe con las tareas que me han sido solicitadas

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.42. Permitir al cliente obtener un informe con las tareas que ha solicitado a determinado profesional

Especificación formal:

> Para evitar ser engañado por un profesional respecto de tareas realizadas pero no pagadas
>
> Como cliente

Quiero que el sistema me permita obtener un informe con las tareas que he solicitado a determinado profesional

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.43. Permitir al profesional obtener un informe con las tareas que le han sido solicitadas por determinado cliente

Especificación formal:

Para evitar ser engañado por un cliente respecto de tareas pagadas pero no realizadas

Como profesional

Quiero que el sistema me permita obtener un informe con las tareas que me ha solicitado determinado cliente

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.44. Permitir al cliente actualizar la valoración dada al trabajo realizado por un profesional

Especificación formal:

Para poder corregir valoraciones que no reflejan lo acaecido con el paso del tiempo

Como cliente

Quiero que el sistema me permita actualizar la valoración dada al trabajo realizado por un profesional

Prioridad asignada: 3

Efectivamente implementado: Solo en el servidor

5.2.45. Permitir al cliente obtener un informe con el historial de valoraciones de un profesional

Especificación formal:

Para poder escoger al profesional adecuado

Como cliente

Quiero que el sistema me permita obtener un informe con el historial de valoraciones de un profesional

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.46. Permitir al cliente obtener un informe con la valoración media de un profesional

Especificación formal:

Para poder escoger al profesional adecuado

Como cliente

Quiero que el sistema me permita obtener un informe con la valoración media de un profesional

Prioridad asignada: 4

Efectivamente implementado: Solo en el servidor

5.2.47. Permitir al profesional obtener un informe con su historial de valoraciones recibidas

Especificación formal:

Para saber qué opinan los clientes sobre mi trabajo

Como profesional

Quiero que el sistema me permita obtener un informe con mi historial de valoraciones recibidas

Prioridad asignada: 3

Efectivamente implementado: No

5.2.48. Permitir al profesional obtener un informe con su valoración media

Especificación formal:

Para saber qué opinan los clientes sobre mi trabajo

Como profesional

Quiero que el sistema me permita obtener un informe con mi valoración media

Prioridad asignada: 3

Efectivamente implementado: Solo en el servidor

5.2.49. Permitir a los profesionales obtener un informe con los productos Premium a la venta

Especificación formal:

Para comprobar si me interesa alguno de los productos Premium a la venta

Como profesional

Quiero que el sistema me permita obtener un informe con los productos Premium a la venta

Para tentar a los profesionales a que compren alguno de los productos Premium a la venta

Como propietario

Quiero que el sistema permita a los profesionales obtener un informe con los productos Premium a la venta

Prioridad asignada: 5

Efectivamente implementado: No

Aclaración: a pesar de su alta prioridad, este requerimiento no ha sido implementado porque falta por definir el contenido de los productos Premium.

5.2.50. Permitir a los profesionales la adquisición de alguno de los productos Premium que los hará aparecer más arriba y les otorgará un distintivo en los informes de profesionales cercanos

Especificación formal:

> **Para** poder beneficiarme de un producto Premium
>
> Como profesional
>
> Quiero que el sistema me permita adquirirlo
>
> **Para** generar ingresos
>
> Como propietario
>
> Quiero que el sistema permita a los profesionales la adquisición de los productos Premium a la venta

Prioridad asignada: 5

Efectivamente implementado: No

Aclaración: a pesar de su alta prioridad, este requerimiento no ha sido implementado porque falta por definir el contenido de los productos Premium.

5.3. Requerimientos no funcionales

En esta sección, para cada requerimiento no funcional obtenido aplicando desarrollo guiado por el comportamiento, detallo la siguiente información:

1. En primer lugar, la especificación formal del requerimiento siguiendo el formato propuesto por BDD.
2. En segundo lugar, el significado del requerimiento no funcional en el contexto de nuestro proyecto.

Además, en el caso de algunos requerimientos añado una reflexión profunda sobre su valor e importancia.

5.3.1. Auditoría y control

Especificación formal:

> Para poder detectar con precisión el origen de los problemas por los que me reclaman los usuarios
>
> Como soporte al cliente
>
> Quiero poder contar con la información necesaria para poder detectar el origen del problema

En el contexto de nuestro proyecto:

El sistema debería garantizar un cierto grado de auditoria y control almacenando eventos preseleccionados, de modo que permita tener control en la parte del sistema correspondiente a la mensajería instantánea y a los pagos.

5.3.2. Eficiencia

Especificación formal:

Para poder ahorrar en costes

Como propietario

Quiero el sistema haga un uso eficiente de los recursos

En el contexto de nuestro proyecto:

El sistema no debe disparar el consumo de ningún recurso para las cargas relativamente bajas a las que lo someteremos.

5.3.3. Eficacia

Especificación formal:

Para proporcionar un servicio de calidad

Como propietario

Quiero que el sistema se mantenga eficaz aún en momentos de esfuerzo

En el contexto de nuestro proyecto:

La relación entre rendimiento y esfuerzo debe mantenerse relativamente constante para los esfuerzos relativamente bajos a los que someteremos el sistema.

5.3.4. Mantenibilidad

Especificación formal:

> Para evitar graves problemas que pueden incluso amenazar la continuidad del negocio
>
> Como propietario
>
> Quiero que el código se mantenga mantenible

En el contexto de nuestro proyecto:

El código debe ser mantenible a toda costa.

Sobre la importancia de la mantenibilidad

El sistema informático inmantenible por excelencia es el que contiene código espagueti que ha echado raíces. El sistema que contiene dicho código puede ser de tres tipos: de soporte genérico al sistema, de soporte específico al sistema y central o nuclear.

Por un lado, si este sistema inmantenible es de soporte genérico al negocio, probablemente la mejor opción sea comprar una solución desarrollada por terceros que desempeñe una función equivalente, o que pueda desempeñarla con la configuración adecuada, y deshacernos del código inmantenible.

Por otro lado, si este sistema inmantenible es de soporte específico al negocio, la mejor opción es rehacerlo si se siguen realizando modificaciones sobre él, o aislarlo tras una API bien

definida si es un sistema legado que ya nadie modifica. En todo caso rehacer todo el código es un proceso muy costoso, mejor hubiera sido hacerle un lavado de cara importante antes de que echara raíces.

Por último, si este sistema inmantenible es central al negocio, pone en peligro la misma continuidad del negocio. Sobre los sistemas centrales al negocio siempre se van a aplicar numerosas mejoras, pues son los que dan valor al negocio y los que hay que renovar constantemente para estar a la última y no ser barridos por la competencia. En el momento en el que el sistema se convierta en inmantenible, los cambios comenzarán a introducir bugs. El equipo que siempre se ha dedicado al sistema seguirá implementando las mejoras, por lo que para los bugs se formará otro equipo, contratando personal si fuera necesario. Este nuevo equipo no conoce el sistema y se le asigna la corrección de un código indescifrable. Con tiempo, el nuevo equipo será capaz de aplicar parches para corregir bugs puntuales, pero surgirán otros. Con más parches y más mejoras sobre el código espagueti arraizado, los bugs van en aumento, de modo que llegado a determinado punto el equipo de operaciones (o de corrección de bugs) es mayor que el equipo que se dedica a implementar las mejoras. Sin embargo, el tamaño adquirido por el equipo de corrección de bugs no los logra erradicar, sino que estos siguen aumentando. Los fallos y los tiempos de espera empiezan a hartar a los clientes. En el momento en que a los propietarios no les valga la pena mantener a semejante equipo para su nivel de clientes en ese momento, la empresa quiebra.

Sobre mi visión sobre este tema para el proyecto

Un sistema cuando alcanza cierto tamaño, al crecer más, su complejidad aumenta más que linealmente con respecto al tamaño del código. Por eso se están popularizando propuestas con

componentes pequeños, como hacer uso de contextos delimitados, componentes de negocio, componentes o microservicios.

Es cierto que la integración de aplicaciones es un asunto que dista mucho de ser trivial. Sin embargo, cuando esta lógica no es muy compleja, la suma de la complejidad de estos pequeños componentes más la complejidad de la lógica de integración suele ser inferior a la complejidad de un componente de gran tamaño equivalente. De esto se deduce que la complejidad del todo no es igual a la suma de las complejidades de las partes.

Es por este motivo y por mi curiosidad insatisfecha con respecto a la integración de sistemas, en este proyecto se optará por una arquitectura de componentes pequeños, si bien no tan pequeños como para poder considerarla una arquitectura de microservicios. En cualquier caso, este asunto está explicado en detalle en el apartado de Arquitectura más adelante en este documento.

5.3.5. Rendimiento/Tiempo de respuesta

Especificación formal:

> Para que se cierren el mayor número de contratos de obra o servicio posibles
>
> Como propietario
>
> Quiero que el sistema de mensajería sea lo más rápido posible.

En el contexto de nuestro proyecto:

Se garantizará un rendimiento mínimo aceptable en el caso de uso que hace un consumo más intensivo de recursos del sistema: el de búsqueda de profesionales cercanos.

5.3.6. Compatibilidad con plataformas

Especificación formal:

Para poder aumentar la base usuarios

Como propietario

Quiero que mi sistema sea compatible con otras plataformas populares

En el contexto de nuestro proyecto:

En el futuro se añadirán nuevos clientes al cliente móvil Android actual. Es fácil garantizar este requerimiento si se publica una API HTTP.

5.3.7. Escalabilidad

Especificación formal:

Para no morir de éxito

Como propietario

Quiero que mi sistema pueda ser escalado de forma sencilla y eficaz

En el contexto de nuestro proyecto:

La escalabilidad es un requerimiento para el caso de uso que hace un consumo más intensivo de los recursos del sistema: el de búsqueda de profesionales cercanos.

Sobre la escalabilidad

Si bien la escalabilidad es un tema que trato en detalle más adelante, está claro que tener pequeños componentes da mucha flexibilidad tanto para escalar verticalmente, como horizontalmente, un componente o varios. También tenemos más maniobra para seleccionar la infraestructura y el hardware más adecuados para cada componente. Otra ventaja.

5.3.8. Testabilidad

Especificación formal:

Para que se testee bien a fondo el código de la aplicación y evitar bugs

Como propietario

Quiero que mi sistema sea altamente testeable

En el contexto de nuestro proyecto:

Es imprescindible que el código sea testeable, independientemente de que después los tests se escriban antes o después de código.

Sobre la testabilidad

Spring que es la tecnología seleccionada para desarrollar la aplicación es un framework ya concebido para hacer el código mucho más testeable. Como contenedor de inversión del control que es, permite componentes muy desacoplados y muy fáciles de mockear.

5.3.9. Usabilidad

Especificación formal:

Para que mi aplicación sea usada fácilmente por prácticamente todo el mundo

Como propietario

Quiero que mi aplicación sea usable

En el contexto de nuestro proyecto:

Es imprescindible que el sistema sea usable.

Sobre la usabilidad

Para lograr una buena usabilidad es imprescindible tener interiorizado y asimilado el modelo conceptual subyacente. Con este conocimiento, probablemente pondremos elementos del modelo fuertemente relacionados en la misma pantalla, y elementos del mismo nivel de granularidad se mostrarán en pantallas a la misma profundidad con respecto a la pantalla de inicio o raíz.

Pongamos un ejemplo del primer caso: cuando un cliente accede al perfil de un profesional, se muestran las valoraciones recibidas por ese profesional de más recientes a más lejanas en el tiempo. Evidentemente, un profesional y las valoraciones que recibe son elementos fuertemente relacionados. Otro ejemplo: cuando el cliente abre el chat o conversación con un determinado profesional, se muestran los mensajes enviados y recibidos ordenados por fecha, de más recientes a más lejanos. Es evidente que el chat entre un cliente y un profesional está fuertemente relacionado con los mensajes que el cliente y el profesional se envían entre ellos.

Pongamos ahora un ejemplo del segundo caso: en la pantalla de inicio de la aplicación se muestran un mapa y un listado de chats como fragmentos alternativos (por defecto se muestra el mapa, siempre que la caché de la aplicación no entre en juego). El mapa representa el contexto delimitado Geodirectorio de Profesionales (componente de alto nivel de abstracción), mientras que el listado de chats representa el contexto delimitado Mensajería Instantánea (componente del mismo alto nivel de abstracción). Los elementos del mapa son Profesionales (entidades, de bajo nivel de abstracción), mientras que los elementos del listado de chats son Chats (entidades, de bajo nivel de abstracción). Para acceder a un profesional o un chat en concreto, lo que hay que hacer es pulsar sobre él, lo cual nos llevará en cualquiera de los dos casos a una pantalla a nivel 1 de profundidad con respecto a la pantalla de inicio con todos los detalles de la entidad en cuestión (profesional o chat, respectivamente).

Para lograr una buena usabilidad, además del mencionado buen conocimiento del dominio se han seguido una serie de directrices:

-En primer lugar, las funcionalidades más populares están más accesibles. Entre las funcionalidades igual de populares, las funcionalidades que se usarán antes están más accesibles. Por ejemplo, buscar profesionales cercanos e iniciar chat con un profesional cercano son las dos funcionalidades muy populares. Sin embargo, no se puede iniciar chat con un profesional cercano si antes no hemos obtenido la lista de profesionales cercanos. De modo que buscar profesionales cercanos está más accesible (un toque) que iniciar chat con un profesional cercano (dos toques, el de la búsqueda y el del inicio del chat).

-En segundo lugar, usar patrones de navegación intuitivos y muy sencillos. Por ejemplo, seleccionar una pestaña abre una pantalla con una lista, y seleccionar un elemento de dicha lista abre una pantalla con los detalles del elemento seleccionado.

-En tercer lugar, hacer uso de la disposición espacial en horizontal de múltiples pantallas, efecto “pestañas”, en la pantalla de inicio. La pantalla de inicio contiene un selector con múltiples opciones, cada una de las cuales te lleva a una pantalla diferente. Mientras navegas entre las opciones del selector, el selector es visible, por lo que todas las pestañas están a distancia 1. Si, en cambio, una vez dentro de una de esas opciones tuviéramos que regresar a la pantalla inicial, y desde allí navegar a otra de las opciones, la distancia sería 2. Al aumentar la anchura del grafo de navegación entre pantallas, disminuye su profundidad. El patrón se puede aplicar recursivamente, de modo que las demás pantallas, además de la inicial, también tengan su propio selector. Sin embargo, en la práctica no se le da este uso, puesto que más de un selector es antiintuitivo y la complejidad se dispara a partir del segundo selector.

-En cuarto lugar, el diseño debe estar libre de distracciones. Esto quiere decir no mostrar información inconexa, pero también evitar información redundante o innecesaria.

5.3.10. Calidad

Especificación formal:

> Para evitar que los usuarios contacten entre ellos por otros medios que no sean la aplicación
>
> Como propietario
>
> Quiero que la parte de mensajería de la aplicación tenga la suficiente calidad como para que los usuarios no tengan que usar aplicaciones de mensajería alternativas.

En el contexto de nuestro proyecto:

Es imprescindible que nuestro sistema tenga un mínimo de calidad, por lo menos la suficiente como para no comprometer la consecución de otros requerimientos no funcionales.

5.3.11. Seguridad y privacidad

Especificación formal:

Para evitar cuantiosas demandas

Como propietario

Quiero que las transacciones dentro de la aplicación sean seguras

Para evitar cuantiosas demandas

Como propietario

Quiero que los credenciales de pago de los usuarios permanezcan ocultos.

Para evitar problemas legales

Como propietario

Quiero evitar exponer la privacidad de mis usuarios

En el contexto de nuestro proyecto:

A pesar de que sin duda es el tema más delicado de cara a un posible lanzamiento de la aplicación, en la fase actual no nos hemos centrado en él.

Sobre la seguridad y la privacidad en nuestra aplicación

Como se ha dicho, este es el tema más delicado antes de salir al mercado. Diría más: este es el asunto más peliguado antes de permitir cualquier uso público de la aplicación móvil, incluso cuando este uso público está limitado, como es el caso de una beta.

Las APIs públicas están y estarán al menos durante una temporada, pero no constituyen ningún peligro. En efecto, los pagos se ejecutan dentro del sandbox de PayPal, y los datos personales no se los estamos pidiendo a nadie.

Si en algún momento se nos ocurriera pensar a futuro en una prueba pública de la aplicación móvil, ya nos encargaríamos de buscar a un consultor experto en seguridad para verificar que cumplimos todos los estándares en seguridad, y que respetamos todas las leyes de protección de datos.

El caso es que el objetivo del proyecto es, como se ha dicho antes, la implementación de un prototipo funcional de uso privado, libre de preocupaciones sobre seguridad o privacidad.

Es precisamente por este motivo que en este texto no se encontrarán referencias ni al Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago, ni a la Ley Orgánica de Protección de Datos, ni a otras.

5.4. Casos de uso

Este sistema dispone de pocos casos de uso. Esto es así porque nuestro sistema está más orientado a eventos, a mensajería y a planificadores que a comandos o consultas.

Sobre los casos de uso y los casos de uso de negocio

Por un lado, los casos de uso determinan la API del servicio, puesto que para cada uno de ellos habrá al menos una subrutina definida en dicha API. En nuestro caso, que definiremos las APIs de todos nuestros sistemas en Java, en vez de hablar de subrutinas hablaremos de métodos.

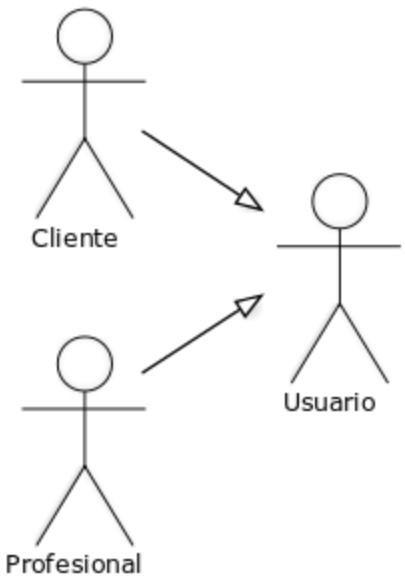
Por otro lado, cada vez leo más sobre los denominados casos de uso de negocio, sobre todo en el contexto de segregación de responsabilidades en comandos y consultas o Command Query Responsibility Segregation (CQRS). Para entendernos, los comandos modifican el estado del sistema, las consultas no; los comandos no reciben ningún informe sobre el estado del sistema, las consultas sí. Los casos de uso de negocio se refieren exclusivamente a aquellos casos de uso que son comandos.

Justificar la elección de esta expresión, casos de uso de negocio, ya es más difícil. En una arquitectura CQRS tenemos dos modelos: el modelo de escritura (sobre el que se ejecutarán los comandos) y el modelo de lectura (sobre el que se informará a las consultas). El estado del modelo de lectura muchas veces no es más que una proyección del estado del modelo de escritura. Por lo tanto, el modelo de lectura es totalmente sustituible por otro, incluso podemos tener varios modelos de lectura para diferentes clientes. Lo que sigue es que lo que es realmente valioso para el negocio es el modelo de escritura, es decir, los comandos. Intuyo que

es por eso que solamente los casos de uso que son comandos reciben el nombre de casos de uso de negocio.

5.4.1. Actores

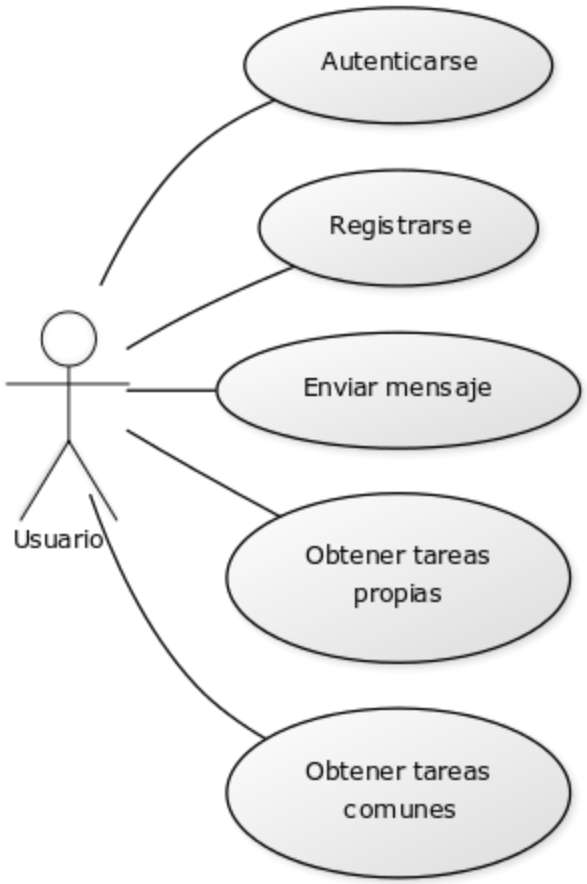
Los actores del sistema son usuarios, quienes a su vez son o bien clientes o bien profesionales.



5.4.2. Diagramas de casos de uso

Para interpretar los casos de uso téngase en cuenta que lo que yo considero sistema es el conjunto del cliente Android, del servidor, y de otros elementos de infraestructura del servidor. Es el siguiente paso, necesario para evitar que la complejidad se dispare, tras considerar en su época el cliente web (el navegador) como parte del sistema.

Por conveniencia y porque está implícito en la jerarquía de actores, no repetiré los casos de uso del usuario ni en el cliente ni en el profesional.



5.4.2.1. Autenticarse

Caso de uso: Autenticarse

Actor primario: Usuario

Postcondiciones:

En caso de éxito: El usuario está autenticado

Precondiciones:

El usuario no está autenticado

Disparadores:

El usuario trata de acceder al sistema sin estar autenticado

Flujo básico:

1. El sistema le muestra al usuario el formulario de autenticación.

2. El usuario escribe sus credenciales

3. El usuario envía sus credenciales al sistema

4. El sistema comprueba que las credenciales sean válidas, y genera y almacena un token de autenticación del usuario

Extensiones:

4.

A. Credenciales no válidas:

El sistema vuelve a ejecutar el paso 1, presentándole al usuario el formulario con las credenciales no válidas e informándole de dicha circunstancia

5.4.2.2. Registrarse

Caso de uso: Registrarse

Actor primario: Usuario

Postcondiciones:

En caso de éxito: El usuario está registrado

Precondiciones:

El usuario no está registrado

Disparadores:

El usuario accede al formulario de registro

Flujo básico:

1. El sistema le muestra al usuario el formulario de registro

2. El usuario completa el formulario

3. El usuario envía el formulario al sistema

4. El sistema comprueba que los datos introducidos por el usuario sean válidos, y los almacena

Extensiones:

2.

A. El usuario cancela el registro

Termina el caso de uso

4.

A. Datos introducidos por el usuario no válidos:

El sistema vuelve a ejecutar el paso 1, presentándole al usuario el formulario con los datos no válidos e informándole de dicha circunstancia

5.4.2.3. Enviar mensaje

Caso de uso: Enviar mensaje

Actor primario: Usuario

Postcondiciones:

En caso de éxito: El usuario ha enviado el mensaje

Precondiciones:

El usuario está autenticado

Disparadores:

El usuario accede a uno de sus chats

Flujo básico:

1. El sistema le muestra al usuario el chat, que contiene un formulario para enviar mensajes

2. El usuario escribe el mensaje

3. El usuario envía el mensaje al sistema

4. El sistema añade el mensaje a la tira de mensajes del chat

5. El sistema envía el mensaje al usuario a quien va destinado

6. El sistema limpia el formulario que sirve para enviar mensajes

Extensiones:

5.4.2.4. Obtener tareas propias

Caso de uso: Obtener tareas propias

Actor primario: Usuario

Postcondiciones:

En caso de éxito: El usuario ha obtenido las tareas propias

Precondiciones:

El usuario está autenticado

Disparadores:

El usuario pide al sistema sus propias tareas

Flujo básico:

1. El sistema busca las tareas del usuario, y se las presenta en la vista correspondiente.

Extensiones:

5.4.2.5. Obtener tareas comunes

Caso de uso: Obtener tareas comunes

Actor primario: Usuario

Postcondiciones:

En caso de éxito: El usuario ha obtenido las tareas comunes con otro usuario

Precondiciones:

El usuario está autenticado

Disparadores:

El usuario pide al sistema las tareas comunes con otro usuario

Flujo básico:

1. El sistema busca las tareas comunes, y se las presenta al usuario en la vista correspondiente.

Extensiones:

5.4.2.6. Especificar ubicación

Caso de uso: Especificar ubicación

Actor primario: Profesional

Postcondiciones:

En caso de éxito: El profesional ha especificado su ubicación

Precondiciones:

El profesional está autenticado

Disparadores:

El profesional accede al formulario para la especificación de su ubicación

Flujo básico:

1. El sistema le muestra al profesional el formulario de ubicación

2. El profesional introduce su ubicación

3. El sistema comprueba que la ubicación sea válida, y la almacena como ubicación del profesional

Extensiones:

2.

A. El profesional cancela la especificación de la ubicación

Termina el caso de uso

3.

B. La ubicación no es válida

El sistema vuelve a ejecutar el paso 1, presentándole al profesional el formulario con la ubicación no válida e informándole de dicha circunstancia

5.4.2.7. Especificar cuenta en la que recibir los pagos

Caso de uso: Especificar cuenta en la que recibir los pagos

Actor primario: Profesional

Postcondiciones:

En caso de éxito: El profesional ha especificado la cuenta en la que desea recibir los pagos

Precondiciones:

El profesional está autenticado

Disparadores:

El profesional accede al formulario para la especificación de la cuenta en la que desea recibir los pagos

Flujo básico:

1. El sistema le muestra al profesional el formulario de cuenta en la que recibir los pagos
2. El profesional introduce la cuenta en la que desea recibir los pagos
3. El sistema comprueba que la cuenta sea válida, y la almacena como cuenta en la que desea recibir los pagos el profesional

Extensiones:

2.

A. El profesional cancela la especificación de la cuenta

Termina el caso de uso

3.

B. La cuenta no es válida

El sistema vuelve a ejecutar el paso 1, presentándole al profesional el formulario con la cuenta no válida e informándole de dicha circunstancia

5.4.2.8. Especificar profesión

Caso de uso: Especificar profesión

Actor primario: Profesional

Postcondiciones:

En caso de éxito: El profesional ha especificado su profesión

Precondiciones:

El profesional está autenticado

Disparadores:

El profesional accede al formulario para la especificación de su profesión

Flujo básico:

1. El sistema le muestra al profesional el formulario de profesión

2. El profesional introduce su profesión

3. El sistema comprueba que la profesión sea válida, y la almacena como profesión del profesional

Extensiones:

2.

A. El profesional cancela la especificación de la profesión

Termina el caso de uso

3.

B. La profesión no es válida

El sistema vuelve a ejecutar el paso 1, presentándole al profesional el formulario con la profesión no válida e informándole de dicha circunstancia

5.4.2.9. Actualizar ubicación

Caso de uso: Actualizar ubicación

Actor primario: Profesional

Postcondiciones:

En caso de éxito: El profesional ha actualizado su ubicación

Precondiciones:

El profesional está autenticado

Disparadores:

El profesional accede al formulario para la actualización de su ubicación

Flujo básico:

1. El sistema le muestra al profesional el formulario de ubicación con su ubicación actual

2. El profesional la sustituye por su nueva ubicación

3. El profesional envía el formulario

4. El sistema comprueba que la nueva ubicación sea válida, y la almacena como ubicación del profesional

Extensiones:

2.

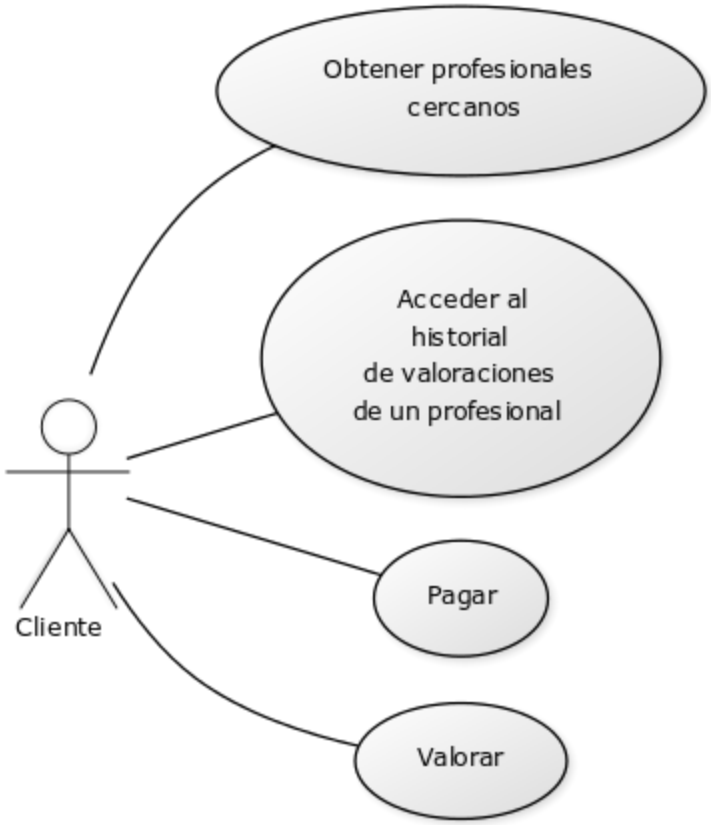
A. El profesional cancela la actualización de la ubicación

Termina el caso de uso

4.

B. La ubicación no es válida

El sistema vuelve a ejecutar el paso 1, presentándole al profesional el formulario con la ubicación no válida e informándole de dicha circunstancia



5.4.2.10. Obtener profesionales cercanos

Caso de uso: Obtener profesionales cercanos

Actor primario: Cliente

Postcondiciones:

En caso de éxito: El cliente ha obtenido sus profesionales cercanos

Precondiciones:

El cliente está autenticado

Disparadores:

El cliente accede al formulario de consulta de sus profesionales cercanos

Flujo básico:

1. El sistema le muestra al cliente el formulario de consulta de sus profesionales cercanos

2. El cliente selecciona una de las profesiones disponibles.

3. El cliente envía el formulario al sistema.

4. El sistema busca los profesionales cercanos al cliente, en su caso de la profesión seleccionada, y se los muestra en la vista correspondiente

Extensiones:

2.

A. El cliente envía el formulario con los valores por defecto

Sigue en el paso 4

B. El cliente cancela la consulta de profesionales cercanos

Termina el caso de uso

5.4.2.11. Acceder al historial de valoraciones de un profesional

Caso de uso: Acceder al historial de valoraciones de un profesional

Actor primario: Cliente

Postcondiciones:

En caso de éxito: El cliente ha obtenido el historial de valoraciones del profesional

Precondiciones:

El cliente está autenticado

Disparadores:

El cliente pide al sistema el historial de valoraciones de un profesional

Flujo básico:

1. El sistema busca el historial de valoraciones del profesional y se lo muestra en la vista correspondiente

Extensiones:

5.4.2.12. Pagar

Caso de uso: Pagar

Actor primario: Cliente

Postcondiciones:

En caso de éxito: El cliente ha efectuado el pago

Precondiciones:

El cliente está autenticado

Disparadores:

El cliente ha indicado al sistema que quiere pagar por la realización de cierta tarea

Flujo básico:

1. El sistema comprueba que dicha tarea no haya sido ya pagada

2. El sistema redirige al cliente a la pasarela de pago

3. La pasarela de pago muestra al cliente el formulario donde debe introducir sus credenciales

4. El cliente escribe sus credenciales

5. El cliente envía sus credenciales a la pasarela de pago

6. La pasarela de pago comprueba que las credenciales del cliente sean válidas

7. La pasarela de pago muestra al cliente el formulario para confirmar el pago

8. El cliente confirma el pago

Extensiones:

1.

A. La tarea ya ha sido pagada

Termina el caso de uso

3, 4, 7

A. El cliente cancela el pago

Termina el caso de uso

6.

A. Las credenciales del cliente no son válidas

El sistema vuelve a ejecutar el paso 3, presentándole al usuario el formulario con las credenciales no válidas e informándole de dicha circunstancia

5.4.2.13. Valorar

Caso de uso: Valorar

Actor primario: Cliente

Postcondiciones:

En caso de éxito: El cliente ha efectuado la valoración

Precondiciones:

El cliente está autenticado

El cliente ha pagado la tarea que va a valorar

Disparadores:

El cliente ha indicado al sistema que quiere valorar la realización de cierta tarea

Flujo básico:

1. El sistema muestra al cliente el formulario de valoración

2. El cliente completa el formulario

3. El cliente envía el formulario

4. El sistema comprueba que los datos del formulario sean válidos y almacena la valoración

Extensiones:

1, 2

A. El cliente cancela la valoración

Termina el caso de uso

6.

A. Las datos del formulario no son válidos

El sistema vuelve a ejecutar el paso 1, presentándole al usuario el formulario con los datos no válidos e informándole de dicha circunstancia

6. Tecnologías

Han sido necesarias muchas y variadas tecnologías para la realización de este proyecto. Para la realización de esta apartado, sin embargo, he realizado una selección y me he quedado solamente con once. Las once tecnologías seleccionadas se han usado durante la fase de implementación, y, o bien se ha hecho un uso intensivo de ellas durante dicha fase, o bien han resultado imprescindibles en la implementación de alguna funcionalidad crítica de la aplicación.

Las tecnologías finalmente seleccionadas han sido las siguientes:

1. La plataforma Java.
2. El framework para Java Spring.
3. La herramienta de mapeo objeto-relacional para Java Hibernate.
4. El sistema gestor de bases de datos MySQL.
5. El sistema gestor de bases de datos PostgreSQL.
6. La extensión para PostgreSQL PostGIS.
7. El servicio de mensajería Firebase Cloud Messaging.
8. El servicio de pagos PayPal Adaptive Payments.
9. El servicio de pagos Braintree Payments.
10. El middleware de mensajería RabbitMQ.

11. El kit de desarrollo software de Android.

Dedicaré una sección a cada una de estas tecnologías. El formato de todas estas secciones será el mismo: en primer lugar, un breve comentario personal sobre la tecnología; en segundo lugar, el motivo de su elección; en tercer lugar, el uso que he hecho de ella en la fase de implementación; y en cuarto lugar, el detalle de la documentación que me ha aportado el conocimiento necesario para hacer un uso competente de la tecnología.

6.1. La plataforma Java

Por plataforma Java me refiero al lenguaje de programación Java, al kit de desarrollo de Java, al entorno de ejecución de Java (con su máquina virtual de Java), y a los múltiples frameworks y a las múltiples librerías disponibles actualmente para Java.

Personalmente, me gusta Java como lenguaje de programación por su orientación a objetos basada en clases, por su tipado estático, por su gestión de la memoria (con el recolector de basura) y por su sintaxis. Sin embargo, existe un lenguaje de programación que prefiero antes que Java: C#.

C# es un lenguaje extremadamente similar a Java: también es orientado a objetos basado en clases, también tiene tipado estático (aunque incluye un tipo dinámico), también gestiona la memoria con el recolector de basura, y la sintaxis a nivel de expresión es idéntica. A pesar del extremo parecido, C# es un lenguaje más completo, en el sentido de que incluye más funcionalidades, y, en general, mejor diseñado, lo cual muchas veces no es visible a nivel de sintaxis pero sí a la hora de compilar.

Dicho lo cual, me quedo con Java, no como lenguaje de programación, sino como plataforma: C# forma parte de la plataforma .NET, la cual es muy difícil de desligar de un entorno Windows y de Microsoft, a pesar de los esfuerzos de los integrantes del proyecto Mono y de otros. Además, C# ya no es un lenguaje de programación tan superior desde que Java dio un gran salto adelante con el lanzamiento de su octava versión, donde incluyó varias de las funcionalidades más importantes que le faltaban con respecto a C#.

La plataforma Java goza de un extraordinario soporte en forma de numerosas librerías y frameworks. De hecho, del resto de tecnologías destacadas en este apartado tenemos que Spring Framework y Hibernate son dos frameworks escritos en Java para Java; que el SDK de Android está escrito en Java para Java (aunque se puede usar C/C++ con el NDK); que Braintree Payments dispone de una pasarela escrita en Java para Java para integrar con ellos; y que RabbitMQ dispone de un cliente escrito en Java para Java.

En cuanto a la documentación, utilicé la novena edición del libro *Java: The Complete Reference* de Herbert Schildt (1312 páginas).

6.2. El framework Spring

La Wikipedia define perfectamente Spring cuando dice que es un framework de aplicaciones y un contenedor de inversión de control para la plataforma Java. Spring pone los componentes (framework) y el flujo (inversión de control). Además, Spring es también excelente a la hora de resolver e inyectar dependencias.

Personalmente, uso Spring por todo lo que implementa por mí, por todo lo que me da hecho. Me ahorra la dificultad de tener que implementarlo por mí mismo, y el tiempo. Además, las

implementaciones que proporciona Spring están muy bien probadas tanto por su equipo de desarrollo como por los miles que lo utilizamos en nuestras aplicaciones. Ese elemento de confianza en la implementación también es importante.

Cuando un framework da mucho hecho, en muchas ocasiones es muy rígido. No ocurre así con Spring. Spring hace uso de convenciones antes que de configuraciones (convention over configuration). Las convenciones, es inevitable, son algo rígidas. Pero uno siempre puede hacer uso de configuración para lograr la flexibilidad necesaria.

La alternativa a Spring es la edición empresarial de Java. Uso Spring porque añade funcionalidades con mucha mayor presteza que la versión empresarial de Java. Esto es algo normal y esperable, puesto que la versión empresarial de Java tiene que adherirse a su propio estándar. En cambio, el estándar, que es lo que me ofrece la edición empresarial de Java, es algo de poco valor para mí.

¿Spring o edición empresarial de Java?

Es sabido que Spring utiliza varias librerías de la edición empresarial de Java: solamente hay que echar un vistazo a sus dependencias. Spring utiliza algunas librerías de la edición empresarial de Java para construir funcionalidad adicional sobre ellas. Spring añade funcionalidades muy deprisa, y eso se ve cuando lanzan una nueva versión de las librerías de la edición empresarial de Java: muchas de las nuevas funcionalidades ya las estaba proporcionando Spring, mientras que muchas de las funcionalidades que ya estaba proporcionando Spring no se han incluido en la nueva versión. Además, las funcionalidades que incluyen las nuevas librerías y que no están disponibles en Spring se añaden a las pocas semanas, y a partir de entonces se empiezan a construir nuevas funcionalidades sobre ellas.

En definitiva, Spring usa librerías de la edición empresarial de Java. Pero eso no es todo: también hay servidores de aplicaciones que se adhieren al estándar de la edición empresarial de Java construidos sobre Spring. De modo que a veces la barrera es difusa.

Spring lo uso para que gestione el flujo de mi aplicación y para que me gestione los componentes y sus dependencias; para hacer uso de sus controladores en la implementación de una API REST; para que, a partir de una sencilla configuración, me provea de una implementación para mis repositorios más simples, para que, junto con Hibernate, se ocupe del mapeo objeto-relacional; para que se encargue de la gestión de las transacciones; y para que me ayude con las pruebas.

A modo de documentación he utilizado la extensa documentación disponible en la página de web de Spring, y la tercera edición del libro Spring Recipes: A Problem-Solution Approach, de Marten Deinum, Daniel Rubio, Josh Long y Gary Mak (828 páginas).

6.3. El framework de mapeo objeto-relacional Hibernate

A pesar de que Hibernate la mayor parte del tiempo lo uso a través de Spring y no directamente, he querido incluirlo en esta sección porque las nueve base de datos de mi aplicación son relacionales, y, en consecuencia, Hibernate está presente, con sus configuraciones, en todos los componentes de mi aplicación.

Personalmente, prefiero configurar Hibernate con XML a hacerlo con anotaciones. Al usar XML, las clases que deben ser mapeadas a tablas en la base de datos quedan mucho más limpias y focalizadas en su objetivo, que, sea el que fuere, no es la persistencia. Esto cobra todavía mayor importancia cuando usamos una arquitectura que aísla el dominio de la infraestructura,

sea cual fuere, porque la inmensa mayoría de clases que se mapean a base de datos forman parte del dominio, y no es deseable que dichas clases tengan configuración de persistencia, puesto que eso es lógica de infraestructura.

Comparemos ahora la versión con anotaciones de persistencia de la clase mensaje de chat, una de las clases del sistema de mensajería instantánea, con la versión sin anotaciones de la misma clase. La clase con anotaciones:

```
@MappedSuperclass
@Table(name = "chat_messages")
public class ChatMessage {
      @EmbeddedId
      @Embedded
      private ChatMessageId id;
      @Embedded
    @AttributeOverride(name="id",column=@Column(name="from_local_id"))
      private UserLocalId from;
      @Embedded
    @AttributeOverride(name="id",column=@Column(name="to_local_id"))
      private UserLocalId to;
      @Embedded
      private ChatMessageBody body;
      @Enumerated(EnumType.STRING)
      private ChatStatus status;

    …
```

```
}
```

Y la misma clase sin anotaciones:

```
public class ChatMessage {

    private ChatMessageId id;

    private UserLocalId from;

    private UserLocalId to;

    private ChatMessageBody body;

    private ChatStatus status;

    …

}
```

Sin duda, la mejora es evidente. A pesar de todo he optado por la configuración con anotaciones porque es más simple.

Hibernate es el framework de mapeo objeto-relacional por excelencia de Java, de modo que la elección ha sido fácil.

Por último, a modo de documentación he utilizado el libro la segunda edición del libro *Java Persistence with Hibernate*, de Christian Bauer, Gavin King y Gary Gregory (608 páginas).

6.4. MySQL

MySQL es un sistema gestor de bases de datos relacionales.

Personalmente, la he escogido porque es gratuito, y porque es fácil de usar y de administrar. Lo que ofrece es suficiente para la inmensa mayoría de componentes de la aplicación, excepto para el geodirectorio de profesionales cercanos, que necesita una base de datos de mayor rendimiento y que permita operar tipos geográficos.

A modo de documentación he utilizado la documentación de la página de MySQL.

6.5. PostgreSQL

PostgreSQL es otro sistema gestor de bases de datos relacionales. De entre los gratuitos, es el mejor en transaccional. De entre todos, Oracle Database está reconocido como el mejor en transaccional.

Personalmente, lo escogí específicamente para el componente geodirectorio de profesionales. Por un lado, necesitaba la base de datos gratuita de mayor rendimiento; por otro lado, necesitaba una base de datos con extensión para sistemas de información geográficos. PostgreSQL me proporcionaba ambas cosas.

A modo de documentación he utilizado la documentación disponible en la página de PostgreSQL.

6.6. La extensión PostGIS

PostGIS es una extensión de PostgreSQL para sistemas de información geográficos.

Personalmente, elegí esta extensión porque al combinarla con PostgreSQL me permite obtener la base de datos geográfica de mayor rendimiento y más completa de entre las gratuitas, lo cual era indispensable para el componente geodirectorio de profesionales de la aplicación.

A modo de documentación he utilizado la segunda versión del libro *PostGIS in Action*, de Regina O. Obe y Leo S. Hsu (600 páginas).

6.7. Firebase Cloud Messaging

Firebase Cloud Messaging, anteriormente Google Cloud Messaging, forma parte del proyecto Firebase de Google.

Personalmente, lo he utilizado para simplificar la implementación del servidor XMPP que permite proveer del servicio de mensajería instantánea. Probablemente, la mayor simplificación la he logrado al delegar en él la gestión directa de las conexiones con los clientes. También lo he elegido por un tema de nombre: el hecho de que esté siendo desarrollado por un equipo de Google ha sido decisivo para decantarme.

A modo de documentación he utilizado la documentación disponible en la página de Firebase que se refería a la parte de Cloud Messaging: tutoriales, referencia y ejemplos.

6.8. El servicio de pagos PayPal Adaptive Payments

PayPal Adaptive Payments es la API de pagos más flexible de PayPal.

Personalmente, escogí esta API porque es sencilla de usar y porque me brindaba la flexibilidad necesaria para configurar los pagos entre usuarios en los que yo me llevaba una comisión y demoraba el cobro por parte del profesional.

A modo de documentación he utilizado la documentación disponible en la página dirigida a desarrolladores de PayPal.

6.9. El servicio de pagos de Braintree Payments

Braintree Payments es un servicio de pagos que, además, proporciona su propio componente de interfaz de usuario insertable tanto en aplicaciones móviles como en páginas web.

Personalmente, lo he escogido por el componente de la interfaz de usuario, que permite introducirlo en tu página web o aplicación sin necesidad de navegar a otra página web o abrir un navegador en el móvil, respectivamente.

A modo de documentación, he utilizado la documentación disponible en la página de Braintree Payments.

6.10. RabbitMQ

RabbitMQ es un middleware orientado a mensajes, es decir, es para la integración de aplicaciones haciendo uso de mensajería.

Personalmente, lo he utilizado por su facilidad de uso y administración. El sistema de intercambiadores y colas que proporciona es más que suficiente para las necesidades de mi aplicación.

A modo de documentación, he utilizado la primera edición, algo obsoleta, del libro *RabbitMQ in Action: Distributed Messaging for Everyone*, de Alvaro Videla y Jason J. W. Williams (312 páginas), además de la documentación disponible en la página web de RabbitMQ.

6.11. El kit de desarrollo software de Android

El kit de desarrollo software para Android proporciona todas las herramientas necesarias para desarrollar aplicaciones en Android.

Personalmente, lo he utilizado porque es la única opción si se quiere desarrollar para Android. Solamente lo he utilizado para el desarrollo del cliente móvil.

A modo de documentación he utilizado la segunda edición del libro Android Programming: The Big Nerd Ranch Guide, de Bill Phillips, Chris Stewart y Brian Hardy (Author) (600 páginas), y la extensa documentación disponible en la página correspondiente para desarrolladores Android.

7. Diseño

En este apartado, lo que haré es lo siguiente:

1. En la primera sección, introduciré el diseño guiado por el dominio.
2. En la segunda sección, elaboraré, aplicando diseño guiado por el dominio, un diseño estratégico que contendrá los componentes del sistema y las relaciones entre ellos.
3. En la tercera sección, elaboraré, aplicando diseño guiado por el dominio, un diseño táctico para cada uno de los componentes obtenidos en la sección anterior.
4. En la cuarta sección, explicaré el estilo arquitectónico aplicado a cada uno de los componentes de este sistema.
5. En la quinta sección, presentaré el diseño del comportamiento mediante diagramas de secuencia explicados.
6. En la sexta sección, presentaré el diseño de los datos mediante diagramas de base de datos explicados.
7. En la séptima sección, expondré un razonamiento profundo para justificar ciertas decisiones de diseño.
8. Por último, en la octava sección, presentaré el diseño de la interfaz de usuario.

7.1. Introducción al diseño guiado por el dominio

7.1.1. Mi definición de diseño guiado por el dominio o Domain-Driven Design (DDD)

DDD es una aproximación al desarrollo software presentada por Eric Evans en su influyente libro *Domain-Driven Design: Tackling Complexity in the Heart of Software*. El subtítulo de este libro, Abordando la Complejidad en el Corazón del Software, da a entender que esta aproximación al desarrollo software fue concebida para lidiar con la complejidad en la parte del software que es más importante.

¿Cuál es la parte del software que es más importante? Depende. Si el software lo está desarrollando una empresa, la parte más importante del software es la que aporte más valor a la empresa. En otros casos, dependerá de lo que se pretenda conseguir con el software.

¿Cómo propone DDD lidiar con esta complejidad? Con una filosofía de desarrollo software y un lenguaje de patrón.

La filosofía de desarrollo software que propone DDD está basada en la colaboración, la comunicación y el contexto. Dado que el equipo humano que está desarrollando este software cuenta con una sola persona, conmigo, aplicar esta filosofía no tiene mucho sentido, de modo que no profundizaré más en ella.

El lenguaje de patrón lo componen dos conjuntos disjuntos de patrones. Cuando digo disjuntos, me refiero a que los patrones de un conjunto, que están relacionados entre ellos, no guardan

relación alguna con los patrones del otro conjunto. Esto es así porque ambos conjuntos están a un diferente nivel de abstracción.

Los dos conjuntos de patrones son los siguientes:

-Los patrones estratégicos: guían el diseño de las relaciones entre diferentes modelos (inter-modelo).

-Los patrones tácticos: guían el diseño de un modelo (intra-modelo).

¿Qué es un modelo? El resultado de modelar un dominio. Cuando el modelado de un dominio resulta en un solo modelo, solamente los patrones tácticos son aplicables. En cambio, cuando el modelado de un dominio resulta en más de un modelo, tanto los patrones tácticos como los patrones estratégicos son aplicables.

Teniendo en cuenta que yo considero que el modelo es el código, un modelo a todos los efectos es lo que en UML se conoce como componente. La definición oficial de componente UML, en inglés, que podemos encontrar en cualquiera de las versiones de la especificación *OMG Unified Modeling Language (OMG UML), Superstructure*, es perfecta, no le sobra una coma:

> A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces.

Que traducido, sería lo siguiente:

> Un componente representa una parte modular de un sistema que encapsula su contenido y cuya manifestación es reemplazable en su entorno. Un componente define su comportamiento en términos de interfaces provistas y requeridas.

Es decir, un componente es la implementación de un servicio. La implementación es el contenido encapsulado y el servicio es la interfaz o interfaces provistas. Un servicio puede consumir otros servicios, para lo cual deberá adaptarse a las interfaces provistas por estos.

Como tal, un componente es totalmente reemplazable por cualquier otro componente que implemente el mismo servicio.

Todo lo que se ha dicho sobre el componente UML es aplicable al modelo DDD.

7.1.2. ¿Modelo o contexto delimitado?

En los textos que versan sobre DDD, las alusiones a los contextos delimitados o bounded contexts son muy frecuentes. Estos contextos delimitados no son más que modelos cuyo contexto ha sido delimitado.

Para los que practicamos DDD, es muy importante que un modelo, aunque evolucione, modele siempre la misma cosa. Es decir, para nosotros es muy importante que el dominio subyacente no varíe. Para esto se delimita el contexto de un modelo: para que modele siempre la misma cosa.

Aprovecho para aclarar dos cosas:

-En primer lugar, que un modelo vacío, otro en construcción e incompleto y otro más terminado y completo, los tres modelos del mismo dominio, modelan la misma cosa.

-En segundo lugar, que un modelo tiene que ceñirse a su dominio, pero que el dominio no se ciñe a ningún modelo. El dominio es el que es. En los raros casos en que sustituyamos un dominio por otro, habrá que construir un modelo para el nuevo dominio. Por ejemplo, supongamos que una empresa que siempre ha vendido bolígrafos, decide comenzar a vender también papel y otros artículos de papelería. Sustituiremos el dominio Bolígrafos por el dominio Artículos de Papelería, y, en su caso, habrá que construir un modelo para el segundo y nuevo dominio. Suponiendo que existiera un modelo del dominio Bolígrafos, podríamos utilizarlo como modelo base, a partir del cual deberíamos construir, del dominio Artículos de Papelería. Y al hacerlo, al usar el modelo de un dominio como modelo base de otro dominio, estaríamos creando un nuevo modelo, puesto que dos modelos con idéntico contenido pero que modelan cosas diferentes son modelos distintos. En este sentido, hay infinitos modelos vacíos distintos, diferenciados únicamente por el dominio que modelan.

Terminadas las dos aclaraciones, volvamos con los contextos delimitados. Los contextos delimitados son especialmente valiosos en grandes sistemas de información con múltiples modelos en los que trabajan múltiples equipos. Tener definidos con precisión los modelos en los que están trabajando los otros equipos y, sobre todo, tener definido con precisión el modelo en el que está trabajando nuestro equipo, resulta una ayuda decisiva para el éxito de los grandes proyectos software.

La existencia de estas delimitaciones en el contexto de los modelos ayuda a evitar que una misma funcionalidad sea implementada por más de un equipo, y la existencia de estas barreras tan claras facilita en gran medida la integración entre modelos diferentes.

¿Por qué los llaman contextos delimitados en vez de modelos con el contexto delimitado que es lo que son? Por conveniencia, porque es más corto. Cuando estás hablando sobre

contextos delimitados, tener que decir “modelos con el contexto delimitado” cada dos o tres frases resulta largo y tedioso. Lo mismo que tener que verlo escrito dos o tres veces en el mismo párrafo. Tanto en español, “modelo con el contexto delimitado”, como en inglés, “model with a bounded context”, mejor dos palabras que cinco.

Sin embargo, la adopción de esta medida por conveniencia puede dar lugar a confusiones. Cuando hablamos de “contexto delimitado”, puede ser que nos refiramos tanto al contexto delimitado en sí como al modelo cuyo contexto está delimitado. Es parecido a si utilizáramos la misma expresión para referirnos al dominio y al modelo que modela el dominio.

7.1.3. Cerrando el círculo: ¿modelo con el contexto delimitado, o modelo ceñido a su dominio subyacente?

Tras leer los dos apartados anteriores, uno puede pensar que son la misma cosa. Pero no: existen diferencias de matiz.

En la teoría, podemos hablar de modelo de un dominio. En la práctica, debemos hablar de modelo de un dominio contextualizado.

¿Qué da contexto a un dominio? La organización para la que se está modelando, el proyecto en el que se está modelando y el equipo que lo está modelando. Así por ejemplo, el dominio Clientela significará cosas diferentes para empresas diferentes, y dentro de la misma empresa, significará cosas diferentes para equipos diferentes trabajando en productos diferentes.

Por eso se dice aquello de que no hay que modelar la realidad, sino la parte de la realidad relevante en un contexto dado.

7.2. Diseño estratégico guiado por el dominio

En esta sección determinaré los contextos delimitados y las relaciones entre estos aplicando los patrones estratégicos de DDD. El punto de partida serán los requerimientos funcionales que he establecido en el apartado anterior correspondiente.

7.2.1. Los contextos delimitados

A partir de la lista de requerimientos funcionales, vamos a establecer grupos de funcionalidades relacionadas entre sí:

Primer grupo:

- Permitir a los usuarios registrarse.
- Permitir a los usuarios autenticarse.
- Recordar credenciales de los usuarios.
- No permitir el uso del cliente móvil a los usuarios si no están autenticados.
- Permitir a los usuarios recuperar su contraseña.
- Verificación de la cuenta de email facilitada por los usuarios al registrarse.
- Permitir a los clientes obtener un informe con los datos personales de un profesional.
- Permitir a los profesionales obtener un informe con los datos personales de un cliente.

Segundo grupo:

- Permitir a los usuarios facilitar datos de contacto de conocidos a los que desean que el sistema envíe una invitación de su parte.
- Envío de invitaciones a conocidos de usuarios.
- Detectar y registrar cuando un usuario se ha registrado a través de la invitación de otro.
- Recompensar a aquellos usuarios cuyas invitaciones han sido utilizadas por otros usuarios para registrarse. La recompensa consistirá en una participación en la tasa que cobre el sistema a los pagos realizados o percibidos por el usuario invitado.
- Permitir a los usuarios obtener un informe con el total percibido por invitaciones exitosas.

Tercer grupo:

- Permitir al cliente iniciar un chat con un profesional.
- Permitir al cliente el envío de mensajes a un profesional.
- Permitir al profesional el envío de mensajes a un cliente.
- Envío de mensajes a un cliente que le han sido enviados por un profesional.
- Envío de mensajes a un profesional que le han sido enviados por un cliente.
- Notificación al usuario de que el mensaje que ha enviado ha sido recibido por el servidor.
- Notificación al usuario de que el mensaje que ha enviado ha sido entregado al dispositivo de su interlocutor.

- Notificación al usuario de que el mensaje que ha enviado ha sido leído por su interlocutor.
- Permitir al usuario bloquear a otros usuarios con los que ha establecido contacto.
- Permitir al usuario denunciar a otros usuarios con los que ha establecido contacto.
- Desactivación de profesionales que no contestan a los mensajes que les envían los clientes para que no salgan en los informes de profesionales cercanos.
- Permitir a los usuarios eliminar un chat.

Cuarto grupo:

- Permitir al profesional especificar sus datos de ubicación.
- Permitir al profesional actualizar sus datos de ubicación.
- Permitir al profesional especificar su profesión.
- Permitir al cliente obtener un informe con profesionales cercanos.
- Permitir al cliente obtener un informe con profesionales cercanos de determinada profesión.
- Desactivación de profesionales que no contestan a los mensajes que les envían los clientes para que no salgan en los informes de profesionales cercanos.
- Permitir a los profesionales la adquisición de alguno de los productos Premium que los hará aparecer más arriba y les otorgará un distintivo en los informes de profesionales cercanos.

Quinto grupo:

- Permitir al profesional especificar la cuenta en la que desea recibir sus pagos.
- Permitir al cliente obtener un informe con el total cobrado por un profesional a través de la aplicación.
- Permitir al cliente pagar al profesional que le ha realizado una tarea.
- Recompensar a aquellos usuarios cuyas invitaciones han sido utilizadas por otros usuarios para registrarse. La recompensa consistirá en una participación en la tasa que cobre el sistema a los pagos realizados o percibidos por el usuario invitado.

Sexto grupo:

- Permitir a los clientes solicitar una tarea a un profesional.
- Permitir a los profesionales realizar un presupuesto para una tarea solicitada por un cliente.
- Permitir a los clientes rechazar un presupuesto realizado por un profesional.
- Permitir a los clientes aceptar un presupuesto realizado por un profesional.
- Permitir a los profesionales indicar que ya han realizado la tarea que les encomendó el cliente.
- Permitir al cliente pagar al profesional que les ha realizado una tarea.
- Permitir al cliente valorar el desempeño del profesional en la realización de determinada tarea.

- Permitir al cliente obtener un informe con las tareas que ha solicitado.
- Permitir al profesional obtener un informe con las tareas que se le han solicitado.
- Permitir al cliente obtener un informe con las tareas que ha solicitado a determinado profesional.
- Permitir al profesional obtener un informe con las tareas que le han sido solicitadas por determinado cliente.

Séptimo grupo:

- Permitir al cliente valorar el desempeño del profesional en la realización de determinada tarea.
- Permitir al cliente actualizar la valoración dada al trabajo realizado por un profesional.
- Permitir al cliente obtener un informe con el historial de valoraciones de un profesional.
- Permitir al cliente obtener un informe con la valoración media de un profesional.
- Permitir al profesional obtener un informe con su historial de valoraciones recibidas.
- Permitir al profesional obtener un informe con su valoración media.

Octavo grupo:

- Permitir a los profesionales obtener un informe con los productos Premium a la venta.

Noveno grupo:

- Permitir a los profesionales la adquisición de alguno de los productos Premium que los hará aparecer más arriba y les otorgará un distintivo en los informes de profesionales cercanos.

De estos nueve grupos de funcionalidades, podemos determinar los nueve contextos delimitados de que consta nuestro sistema:

1. Identidad y acceso de usuarios.
2. Invitaciones.
3. Mensajería instantánea.
4. Geodirectorio de profesionales.
5. Pagos de tareas.
6. Tareas.
7. Valoraciones de tareas.
8. Catálogo de productos Premium.
9. Venta de productos Premium.

7.2.2. Las relaciones entre contextos delimitados

En primer lugar, lo que haremos es establecer los contextos delimitados de que depende cada uno de los nueve contextos delimitados determinados en el apartado anterior:

1. Identidad y acceso de usuarios: no depende de nadie.

2. Invitaciones: depende de:

    a. Identidad y acceso de usuarios. El contexto delimitado Invitaciones debe informarse de aquellos usuarios que se han registrado haciendo uso de invitaciones enviadas por otros usuarios para recompensar a los invitadores.

    b. Pagos de tareas. El contexto delimitado Invitaciones debe informarse de los pagos de tareas para recompensar al invitador del pagador y al invitador del pagado.

3. Mensajería instantánea: no depende de nadie.

4. Geodirectorio de profesionales: depende de:

    a. Identidad y acceso de usuarios. El contexto Geodirectorio de profesionales debe comprobar que quien modifica los datos de un profesional es el propio profesional para evitar que una persona no autorizada realice estas modificaciones.

    b. Pagos de tareas. El contexto Geodirectorio de profesionales debe informarse de los pagos que recibe un profesional por la realización de tareas para poder calcular sus ganancias totales.

    c. Valoraciones de tareas. El contexto Geodirectorio de profesionales debe informarse de las valoraciones que recibe un profesional por la realización de tareas para poder calcular su valoración media.

    d. Venta de productos Premium. El contexto Geodirectorio de profesionales debe informarse de las adquisiciones de productos Premium que realiza un

profesional para poder mostrarlo más arriba en los informes de profesionales cercanos y para poder otorgarle un distintivo.

5. Pagos de tareas: depende de:

    a. Tareas. El contexto Pagos de tareas debe informarse de los detalles de la tarea que se va a pagar para comprobar por ejemplo que no haya sido ya pagada.

6. Tareas: no depende de nadie.

7. Valoraciones de tareas: depende de:

    a. Tareas. el contexto Pagos de tareas debe informarse de los detalles de la tarea que se va a valorar para comprobar por ejemplo que quien valora es quien solicitó la tarea.

8. Catálogo de productos Premium: no depende de nadie.

9. Venta de productos Premium: depende de:

    a. Catálogo de productos Premium. El contexto Venta de productos Premium debe informarse de los detalles de la tarea para comprobar por ejemplo su precio.

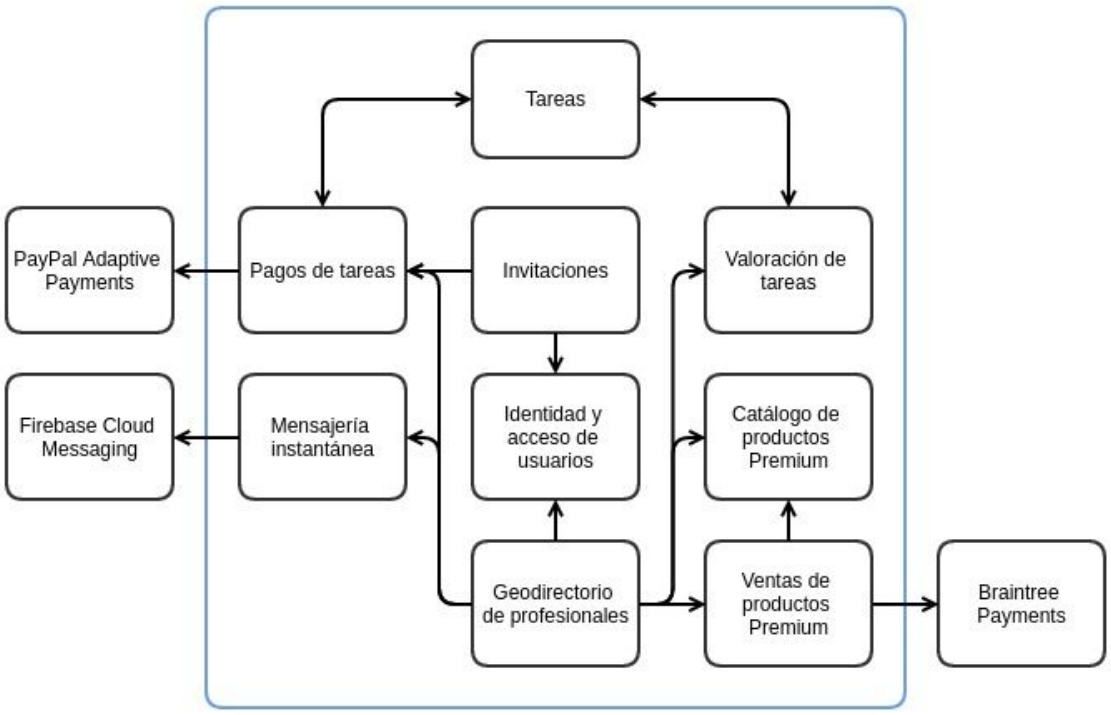
En segundo lugar, lo que haremos es indicar dependencias con sistemas externos.

- Mensajería instantánea: depende de Firebase Cloud Messaging (comunicación vía XMPP).

- Pagos de tareas: depende de PayPal Adaptive Payments (comunicación vía HTTP).

- Venta de productos Premium: depende de Braintree Payments (comunicación vía HTTP).

En tercer lugar, lo que haremos es mostrar las dependencias internas que hemos determinado, y las dependencias externas que hemos indicado, de forma visual:



Los conectores van desde el dependiente hasta la dependencia. Lo que queda fuera de la caja de contorno azul son los sistemas externos.

En cuarto lugar, lo que haremos es explicar los patrones estratégicos de DDD que aplicaremos para determinar las relaciones entre contextos delimitados y sus equipos responsables:

1. Cliente-Proveedor: en esta relación cliente-proveedor, el proveedor tiene en cuenta a la hora de desarrollar las prioridades y las necesidades del cliente. El éxito del sistema proveedor está vinculado al éxito del sistema cliente. Es el caso, por ejemplo, de una organización que está desarrollando internamente una librería específica para una de sus aplicaciones.

2. Conformista: en esta relación cliente-proveedor, el proveedor no tiene en cuenta a la hora de desarrollar las prioridades y las necesidades del cliente. El éxito del sistema proveedor no está vinculado al éxito del sistema cliente. Es el caso, por ejemplo, de un sistema que utiliza un proveedor de servicios de pago pero que factura poco.

En quinto lugar, lo que haremos es aplicar los patrones estratégicos de DDD explicados en el punto anterior para determinar las relaciones entre contextos delimitados y sus equipos responsables:

1. Cliente-Proveedor: cuando uno de nuestros contextos delimitados consume otro síncronamente vía HTTP-REST. Evidentemente, el contexto delimitado consumidor será el cliente, mientras el que consumido será el proveedor.

2. Conformista: cuando uno de nuestros contextos delimitados consume otro asíncronamente vía Eventos-Mensajería; también cuando uno de nuestros contextos delimitados consume un sistema externo. Evidentemente, el contexto delimitado consumidor será el conformista.

El patrón conformista para lograr sistemas autónomos desacoplados

Puede resultar sorprendente que, dentro de una misma organización, establezcamos que algunos equipos deben conformarse con lo que les ofrezcan otros equipos. Esto se hace así

para lograr sistemas autónomos desacoplados. Por un lado, los sistemas que publican eventos no saben qué otros sistemas están suscritos a dichos eventos: publicarán exactamente igual si hay un millar de suscriptores que si no hay ninguno, y publicarán exactamente igual sean quienes sean los suscriptores. Por otro lado, los sistemas que se suscriben a eventos no saben qué sistemas publican dichos eventos: se suscribirán exactamente igual si hay un millar de publicadores que si no hay ninguno, y se suscribirán exactamente igual sean quienes sean los publicadores.

-

En sexto lugar, lo que haremos es explicar los patrones estratégicos de DDD que vamos a aplicar para determinar las relaciones de integración entre contextos delimitados:

1. Servicio alojado en un servidor abierto u Open Host Service (OHS): contexto delimitado o sistema que publica sus servicios para que puedan ser consumidos por todo tipo de clientes.

2. Lenguaje publicado o Published Language (PS): publicado por los contextos delimitados o sistemas OHS para facilitar a los clientes su consumo.

3. Capa anticorrupción o Anticorruption Layer (AL): la implementan los clientes de OHS para adaptar los conceptos del OHS a conceptos del propio modelo, y así evitar corromper el propio modelo. Para la adaptación de conceptos, los clientes harán uso del PS publicado por el OHS.

En séptimo lugar, lo que haremos es aplicar los patrones estratégicos de DDD para determinar las relaciones de integración entre contextos delimitados:

1. OHS con PS: todos nuestros contextos delimitados o sistemas que publican servicios vía HTTP-REST, o que publican eventos vía Mensajería.

2. AL: todos nuestros contextos delimitados o sistemas que consumen OHS con PS propios o sistemas externos.

En octavo lugar, lo que haremos es definir todas y cada una de las relaciones de dependencia entre contextos delimitados:

1. Pagos de tareas depende de Tareas: cuando un cliente se dispone a pagar una tarea, Pagos de tareas consulta los detalles de la tarea a Tareas, detalles como el solicitante de la tarea, el realizador de la tarea y el precio. Pagos de tareas implementaría una AL, mientras que Tareas publicaría el servicio para obtener los detalles de una tarea como OHS con PS.

2. Tareas depende de Pagos de tareas: cuando un cliente confirma el pago de una tarea, Pagos de tareas publica un evento conforme esa tarea ha sido pagada. Dicho evento es consumido por Tareas, el cual actualiza el estado de la tarea a pagado. Tareas implementaría una AL, mientras que Pagos de tareas publicaría el evento como OHS con PS.

3. Valoración de tareas depende de Tareas: cuando un cliente valora la realización de una tarea, Valoración de tareas consulta los detalles de la tarea a Tareas, detalles como el solicitante de la tarea, el realizador de la tarea o el precio de la tarea. Valoración de tareas implementaría una AL, mientras que Tareas publicaría el servicio para obtener los detalles de una tarea como OHS con PS.

4. Tareas depende de Valoración de tareas: cuando un cliente valora una tarea, Valoración de tareas publica un evento conforme esa tarea ha sido valorada. Tareas implementaría una AL, mientras que Valoración de tareas publicaría el evento como OHS con PS.

5. Invitaciones depende de Identidad y acceso de usuarios: cuando un usuario se registra por medio de la invitación de otro usuario, Identidad y acceso de usuarios publica un evento conforme un usuario se ha registrado haciendo uso de una invitación que le fue mandada por otro usuario. Invitaciones registra este hecho y recompensa al invitador. Invitaciones implementaría una AL, mientras que Identidad y acceso de usuarios publicaría el evento como OHS con PS.

6. Invitaciones depende de Pagos de tareas: cuando un cliente confirma el pago de una tarea, Pagos de tareas publica un evento conforme esa tarea ha sido pagada. Invitaciones consume ese evento para hacer partícipes de la tasa que cobra nuestra aplicación al invitador del pagador y al invitador del pagado.

7. Geodirectorio de profesionales depende de Identidad y acceso de usuarios: cuando alguien accede a datos sensibles de un profesional, hay que comprobar que este alguien sea el propio profesional. Lo mismo es aplicable a la actualización de los datos de un profesional: solamente el propio profesional puede actualizarlos. Así pues, cuando un usuario quiere realizar alguna de estas dos acciones, lo que hace Geodirectorio de profesionales es comprobar la identidad de este usuario. Si no es el propio profesional, al usuario se le denegará el acceso. Geodirectorio de profesionales implementaría una AL, mientras que Identidad y acceso de usuarios implementaría un OHS con PS.

8. Geodirectorio de profesionales depende de Pagos de tareas: cuando un cliente confirma el pago de una tarea, Pagos de tareas publica un evento conforme la tarea realizada por el correspondiente profesional ha sido pagada. Geodirectorio de profesionales lo que hace es añadir la cantidad del pago al profesional correspondiente, puesto que almacena para cada profesional su total de ingresos. Geodirectorio de profesionales implementaría una AL, mientras que Pagos de tareas implementaría un OHS con PS.

9. Geodirectorio de profesionales depende de Valoraciones de tareas: cuando un cliente valora el desempeño de un profesional en la realización de cierta tarea, Valoraciones de tareas publica un evento conforme esta valoración se ha realizado. Geodirectorio de profesionales lo que hace es actualizar la valoración media del profesional valorado haciendo uso de la información proporcionada por el evento. Geodirectorio de profesionales implementaría una AL, mientras que Valoraciones de tareas implementaría un OHS con PS.

10. Geodirectorio de profesionales depende de Venta de productos Premium: cuando un profesional adquiere un producto Premium, Venta de productos Premium publica un evento conforme se ha realizado dicha adquisición. Geodirectorio de profesionales utiliza esta información para promocionar el profesional a Premium, y así beneficiarlo en los informes de profesionales cercanos. Geodirectorio de profesionales implementaría una AL, mientras que Venta de productos Premium publicaría el evento como un OHS con PS.

11. Geodirectorio de profesionales depende de Catálogo de productos Premium: cuando a Geodirectorio de profesionales se le notifica que cierto profesional ha adquirido un producto Premium, Geodirectorio de profesionales tiene que informarse acerca de este

producto Premium, a qué tipo de profesional Premium promociona y durante cuánto tiempo. Geodirectorio de profesionales implementaría una AL, mientras que Catálogo de Productos publicaría el servicio de obtención de producto Premium como un OHS con PS.

12. Geodirectorio de profesionales depende de Mensajería instantánea: cuando un usuario no se conecta y no contesta los mensajes, Mensajería instantánea desactiva a este usuario y publica un evento con respecto a esta desactivación. Cuando un usuario que estaba desactivado se conecta, Mensajería instantánea activa a este usuario y publica un evento con respecto a esta activación. Geodirectorio de profesionales usa la información proporcionada por estos eventos para activar o desactivar profesionales, para que aparezcan o no en los informes de profesionales cercanos. Geodirectorio de profesionales implementaría una AL, mientras que Mensajería instantánea publicaría los eventos como un OHS con PS.

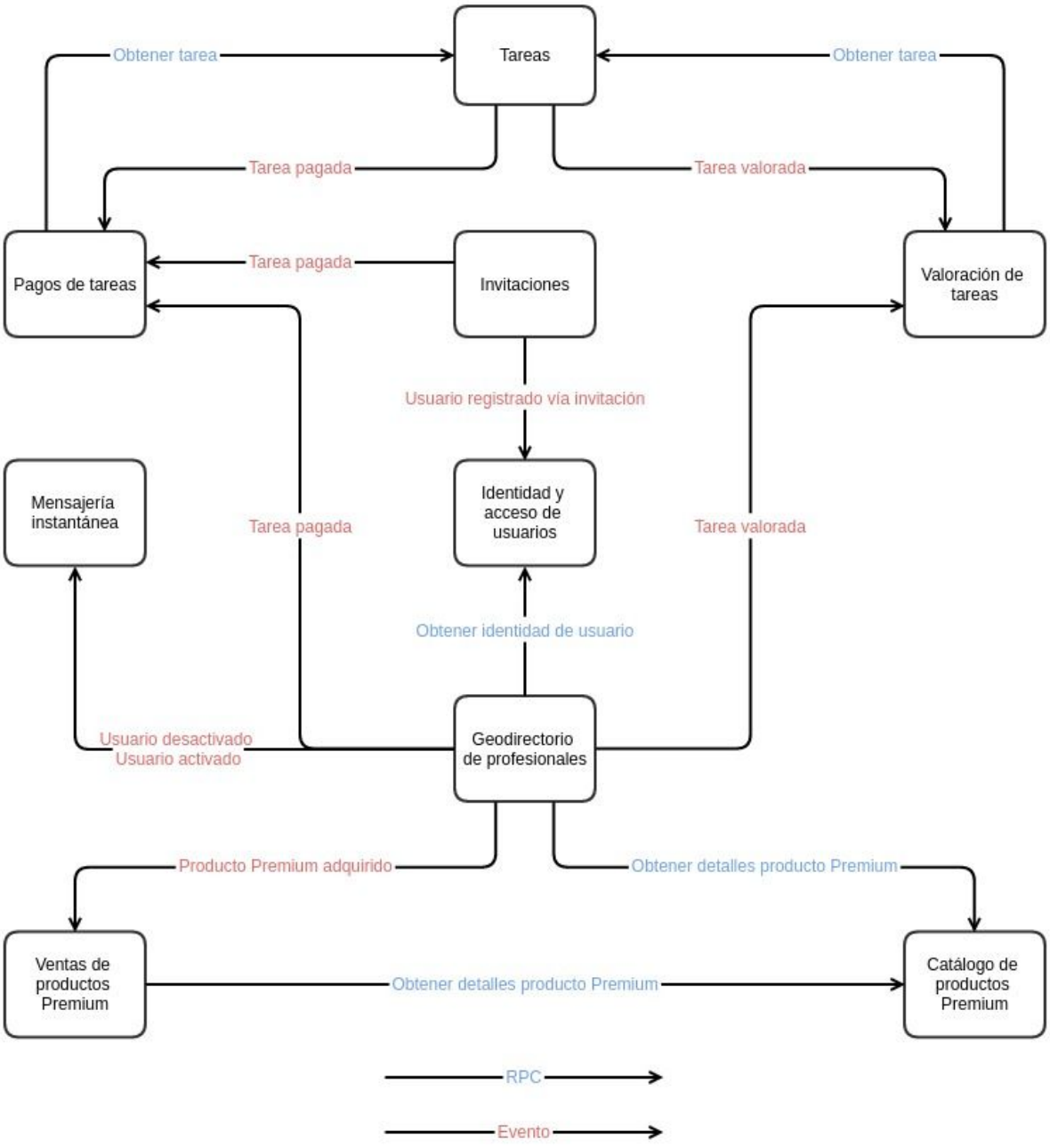
13. Venta de productos Premium depende de Catálogo de productos Premium: cuando un profesional procede al pago de un producto Premium, Venta de productos Premium tiene que informarse acerca de los detalles del producto Premium tales como su precio. Geodirectorio de profesionales implementaría una AL, mientras que Catálogo de productos Premium implementaría el servicio como un OHS con PS.

14. Pagos de tareas depende de PayPal Adaptive Payments: Pagos de tareas implementaría una AL y se comportaría como un Conformista.

15. Venta de productos Premium depende de Braintree Payments: Venta de productos Premium implementaría una AL y se comportaría como un Conformista.

16. Mensajería instantánea depende de Firebase Cloud Messaging: Mensajería instantánea implementaría una AL y se comportaría como un Conformista.

En noveno lugar, lo que haremos es ilustrar todas las integraciones entre contextos delimitados:


Tareas
Obtener tarea
Obtener tarea
Tarea pagada
Tarea valorada
Pagos de tareas
Tarea pagada
Invitaciones
Valoración de tareas
Usuario registrado vía invitación
Mensajería instantánea
Tarea pagada
Identidad y acceso de usuarios
Tarea valorada
Obtener identidad de usuario
Usuario desactivado
Usuario activado
Geodirectorio de profesionales
Producto Premium adquirido
Obtener detalles producto Premium
Ventas de productos Premium
Obtener detalles producto Premium
Catálogo de productos Premium
RPC
Evento

En cuanto a la arquitectura del sistema, por un lado es una arquitectura orientada a servicios o Service Oriented Architecture (SOA), y por otro lado es una arquitectura guiada por eventos o Event Driven Architecture (EDA).

7.3. Diseño táctico guiado por el dominio

En esta sección vamos a diseñar el modelo contenido en varios de los contextos delimitados determinados en la sección anterior:

1. Identidad y acceso de usuarios.
2. Invitaciones.
3. Mensajería instantánea.
4. Geodirectorio de profesionales.
5. Pagos de tareas.
6. Tareas.
7. Valoraciones de tareas.

Para cada uno de estos contextos delimitados, haremos dos cosas:

En primer lugar, establecer los agregados a partir de las funcionalidades y de las reglas de negocio del contexto delimitado.

En segundo lugar, establecer el resto de elementos del contexto delimitado, para obtener el modelo completo.

Breve introducción los patrones tácticos de DDD

- Objeto valor o Value Object: son objetos inmutables, que describen características o propiedades de otros objetos valor o entidades, y que no tienen identidad: dos objetos

valor son iguales si representan el mismo valor. Un String en Java es un objeto valor: inmutable y sin identidad, dos Strings son iguales si representan la misma cadena de caracteres.

- Entidad o Entity: son objetos con identidad explícita, mutables en sus campos no identitarios. Dos objetos entidad son iguales si y solo si sus identidades son iguales. No hay que confundir la identidad modelada explícitamente en las entidades con la identidad implícita que Java por ejemplo asigna a los objetos.

- Agregado o Aggregate: entidad con identidad global y su contenido, que suele ser un agregado de objetos valor y, en ocasiones, entidades con identidad local dentro del agregado. A la entidad con identidad global se la llama Raíz del agregado.

- Eventos: representan la ocurrencia de sucesos significativos. Los publican los agregados.

- Servicios: como diría Eric Evans, “a veces, simplemente no es una cosa”. Los servicios representan lógica de negocio sin estado.

7.3.1. Identidad y acceso de usuarios

7.3.1.1. Los agregados

Determinaremos los agregados a partir de las siguientes funcionalidades:

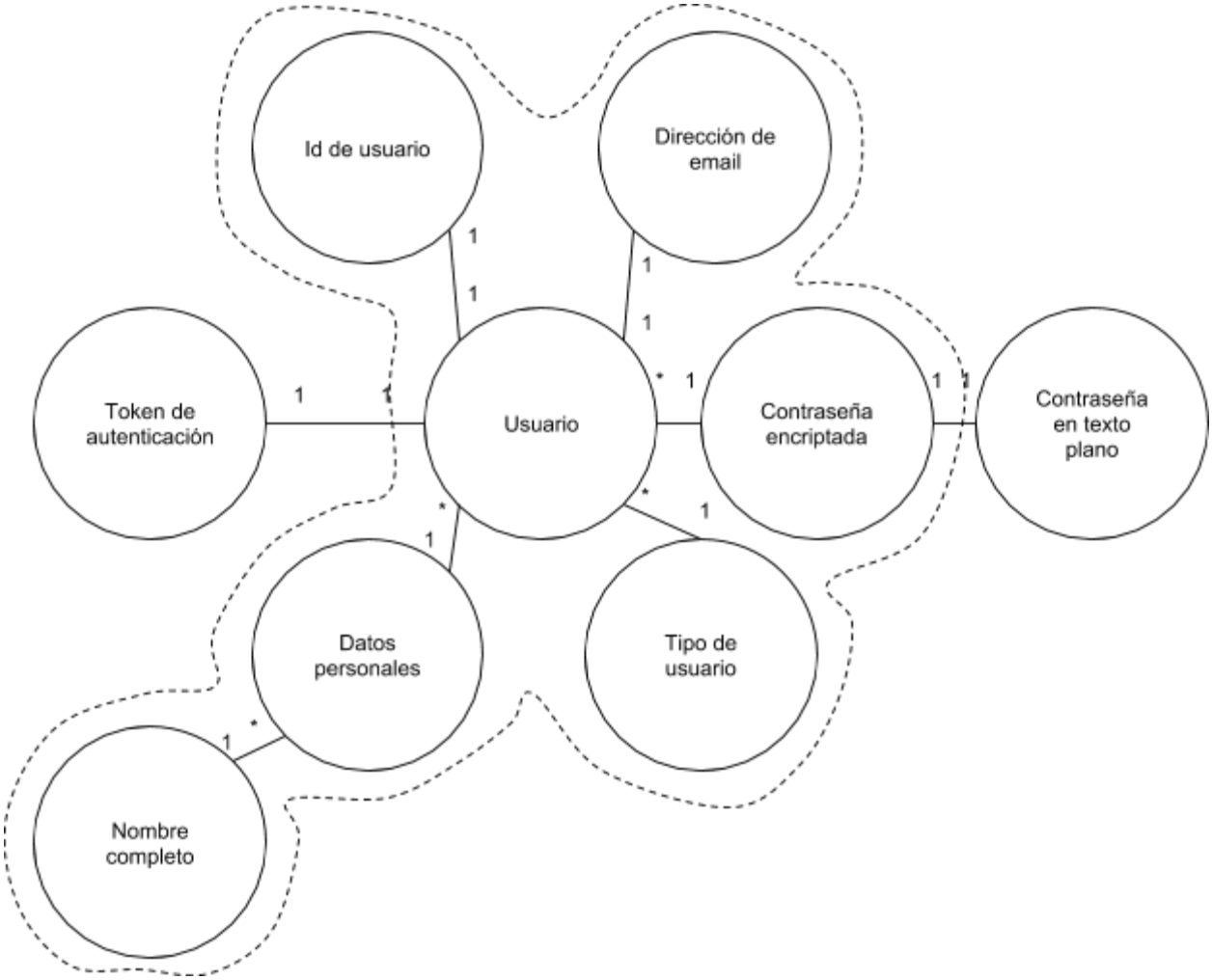
- Permitir a los usuarios registrarse.
- Permitir a los usuarios autenticarse.
- Recordar credenciales de los usuarios.
- No permitir el uso del cliente móvil a los usuarios si no están autenticados.
- Permitir a los clientes obtener un informe con los datos personales de un profesional.
- Permitir a los profesionales obtener un informe con los datos personales de un cliente.

Y a partir de las siguientes reglas de negocio:

- Las credenciales de autenticación de un usuario son su dirección de email y su contraseña.
- La contraseña de un usuario se guardará encriptada.
- Un usuario además contiene sus datos personales.
- Los datos personales del usuario consisten en su nombre completo.
- Cuando se autentique un usuario, se le enviará un token de autenticación que puede incluir en futuras peticiones.

- Los informes con los datos personales de un usuario deben ser una vista simplificada que no incluya información sensible.

De todo ello obtenemos lo siguiente:



De lo que obtenemos:

- 1 agregado: Usuario.

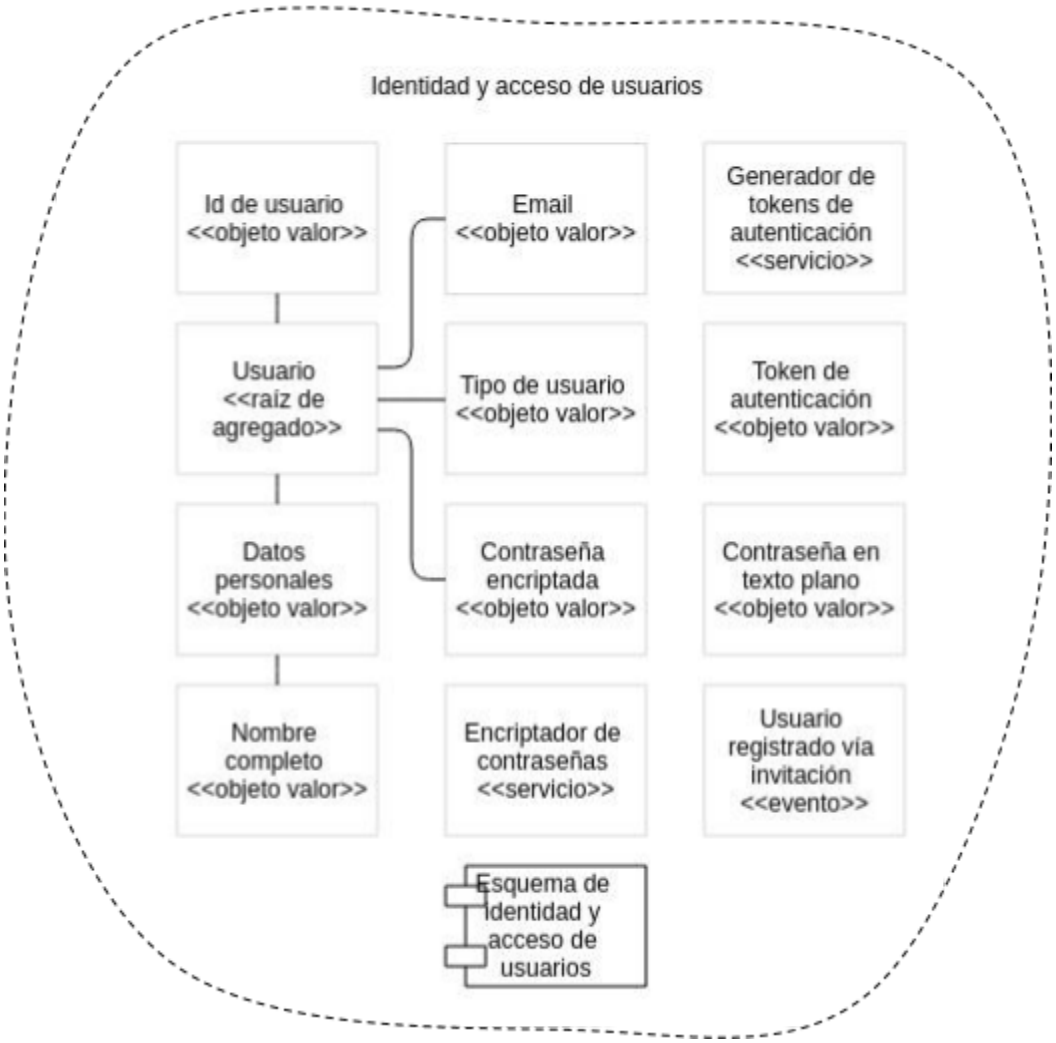
- 2 objetos valor: Token de autenticación y contraseña en texto plano.

7.3.1.2. El modelo

Lo que falta por añadir al modelo:

- 2 servicios: Generador de tokens de autenticación y encriptador de contraseñas.
- 1 evento: Usuario registrado vía invitación.

El modelo definitivo:



7.3.2. Invitaciones

7.3.2.1. Los agregados

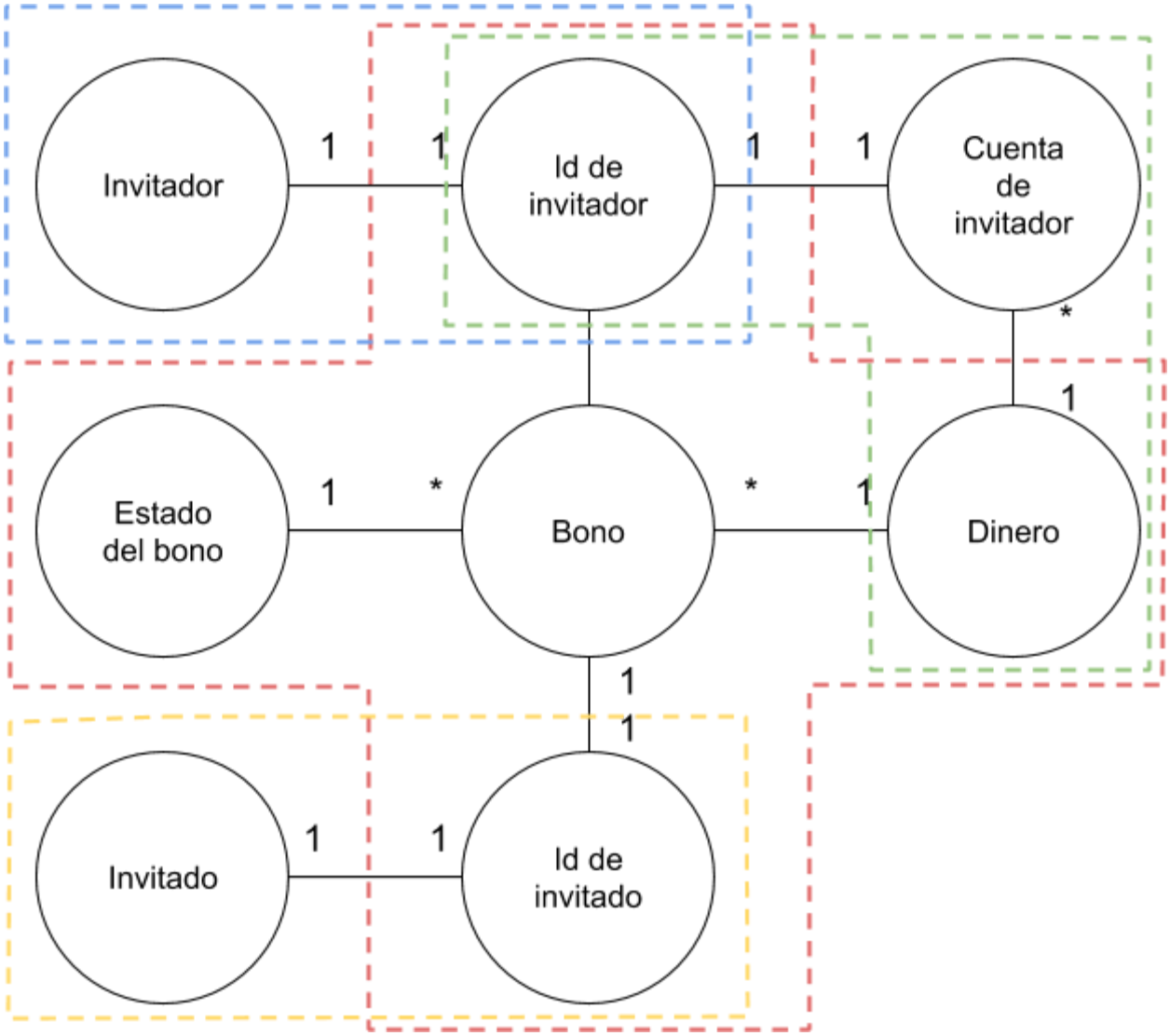
Determinaremos los agregados a partir de las siguientes funcionalidades:

- Detectar y registrar cuando un usuario se ha registrado a través de la invitación de otro.
- Recompensar a aquellos usuarios cuyas invitaciones han sido utilizadas por otros usuarios para registrarse. La recompensa consistirá en una participación en la tasa que cobre el sistema a los pagos realizados o percibidos por el usuario invitado.
- Permitir a los usuarios obtener un informe con el total percibido por invitaciones exitosas.

Y de las siguientes reglas de negocio:

- Toda vez que un usuario se registre por medio de una invitación, al invitador se le premiará con un bono de 100€ que podrá ir cobrando poco a poco a medida que el invitado registrado pague o sea pagado, puesto que hasta el 25% de la tasa que cobre la aplicación en esos pagos le corresponde al invitador.

De todo ello obtenemos lo siguiente:



De lo que obtenemos:

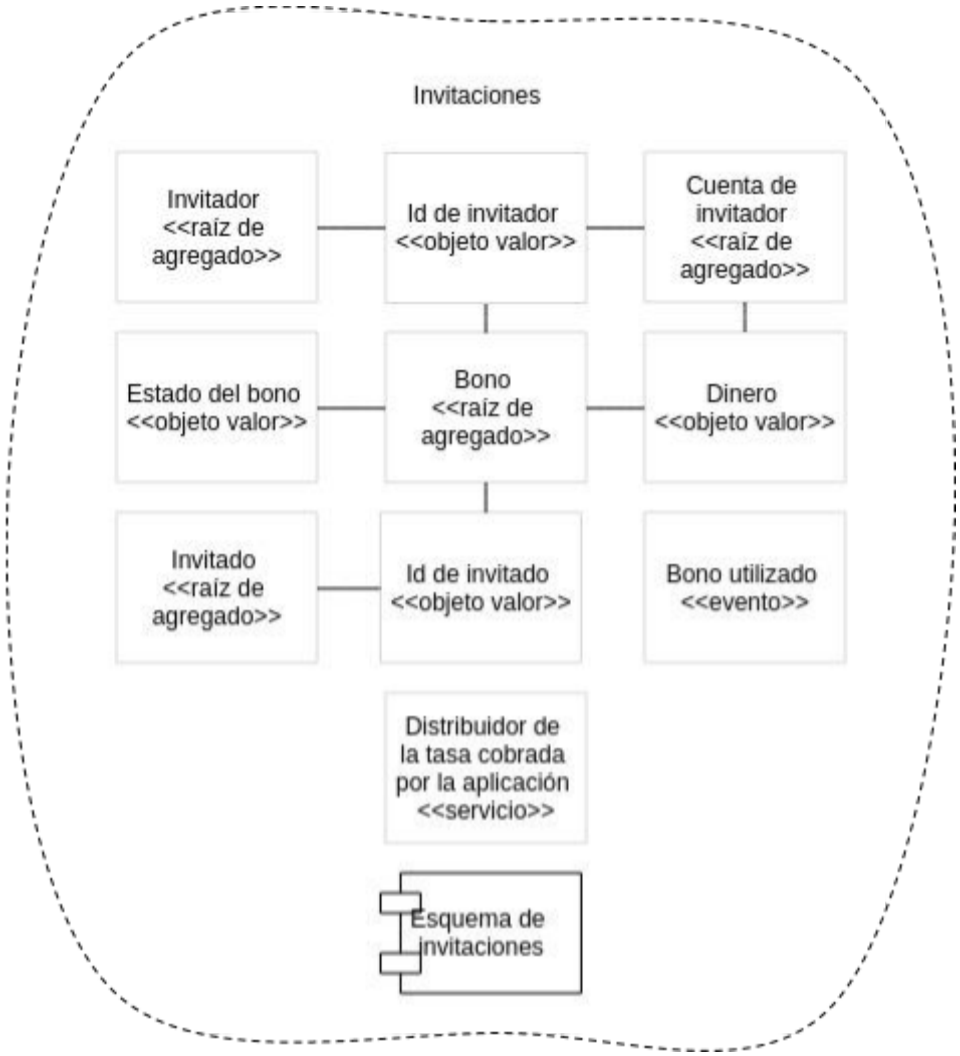
- 4 agregados: Invitador, Invitado, Bono y cuenta de invitador.

7.3.2.2. El modelo

Lo que falta por añadir al modelo:

- 1 servicio: distribuidor de la tasa cobrada por la aplicación.
- 1 evento: bono utilizado.

El modelo definitivo:



7.3.3. Mensajería instantánea

7.3.3.1. Los agregados

Determinaremos los agregados a partir de las siguientes funcionalidades:

- Permitir al cliente el envío de mensajes a un profesional.
- Permitir al profesional el envío de mensajes a un cliente.
- Envío de mensajes a un cliente que le han sido enviados por un profesional.
- Envío de mensajes a un profesional que le han sido enviados por un cliente.

Y de las siguientes reglas de negocio:

- Nuestro sistema de mensajería utiliza un servidor de mensajería intermedio tanto para recibir como para enviar mensajes.
- Dicho servidor de mensajería intermedio tiene su propio sistema de identificación de usuarios.
- Cuando recibimos mensajes de chat, debemos enviárselos a su destinatario haciendo uso del identificador de este en el servidor de mensajería intermedio.
- Cuando le enviamos un mensaje, el servidor de mensajería intermedio puede confirmar el mensaje, o negar la confirmación.
- Cuando niega la confirmación, el servidor de mensajería intermedio especifica una causa. Esta causa determina si el mensaje puede enviarse de nuevo o no, y en el caso

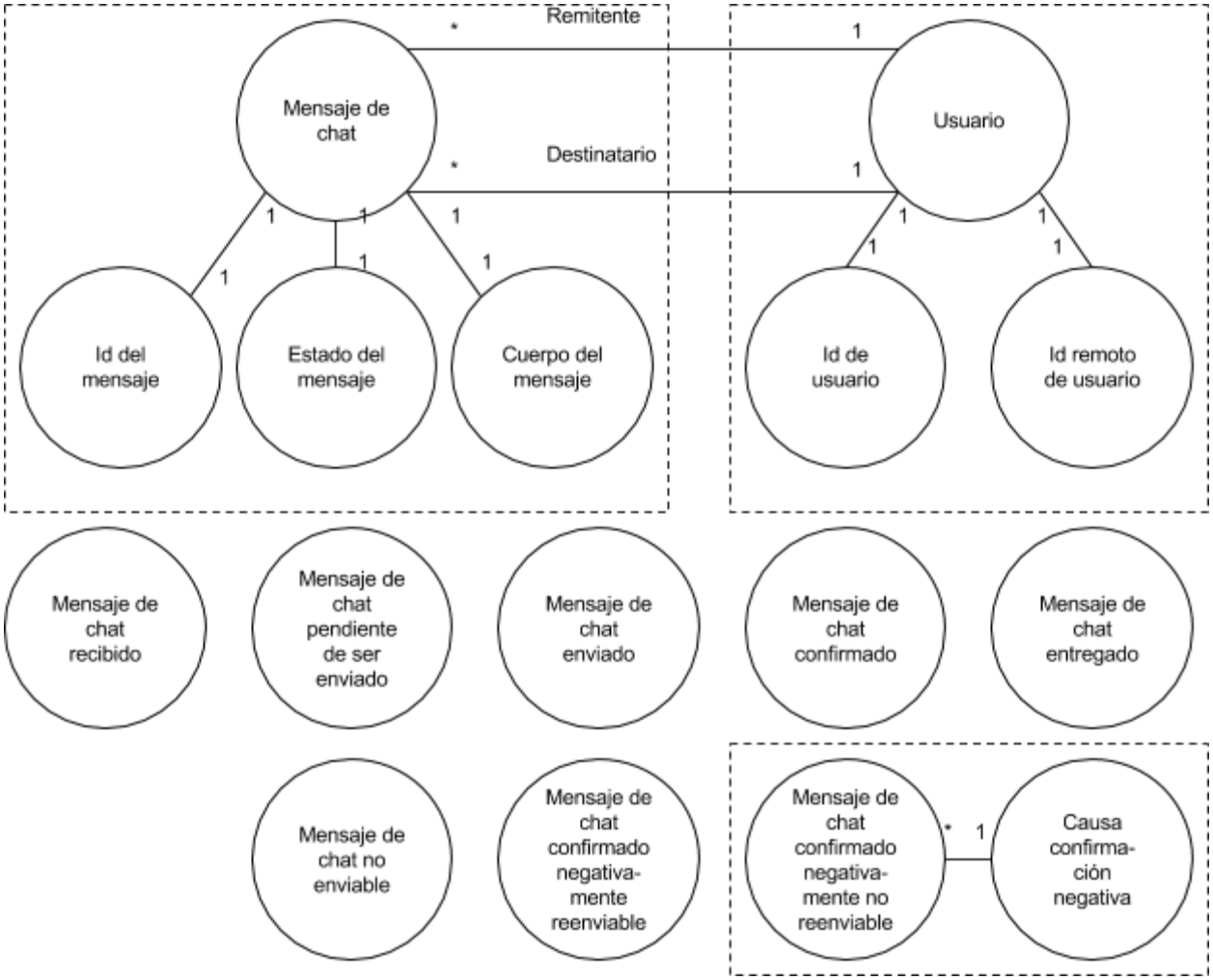
de que el mensaje pueda enviarse de nuevo, esta causa determina si el mensaje puede enviarse inmediatamente o no.

- En el caso de que no proceda reintentar el envío inmediatamente, el tiempo de espera hasta el próximo reintento deberá ser determinado haciendo uso de un algoritmo de backoff exponencial.

- El servidor de mensajería intermedio nos notificará cuando el mensaje llegue al dispositivo de su destinatario final.

Como se puede ver, algunas de estas reglas de negocio incluyen políticas de Firebase Cloud Messaging. Esto es consecuencia de la adopción de un rol conformista con respecto a dicho sistema externo. Podría solucionarse parte de esta fuerte dependencia con Firebase Cloud Messaging moviendo parte de la lógico de aplicación y del dominio a infraestructura, pero no se ha considerado necesario.

De todo ello obtenemos lo siguiente:



De lo que obtenemos:

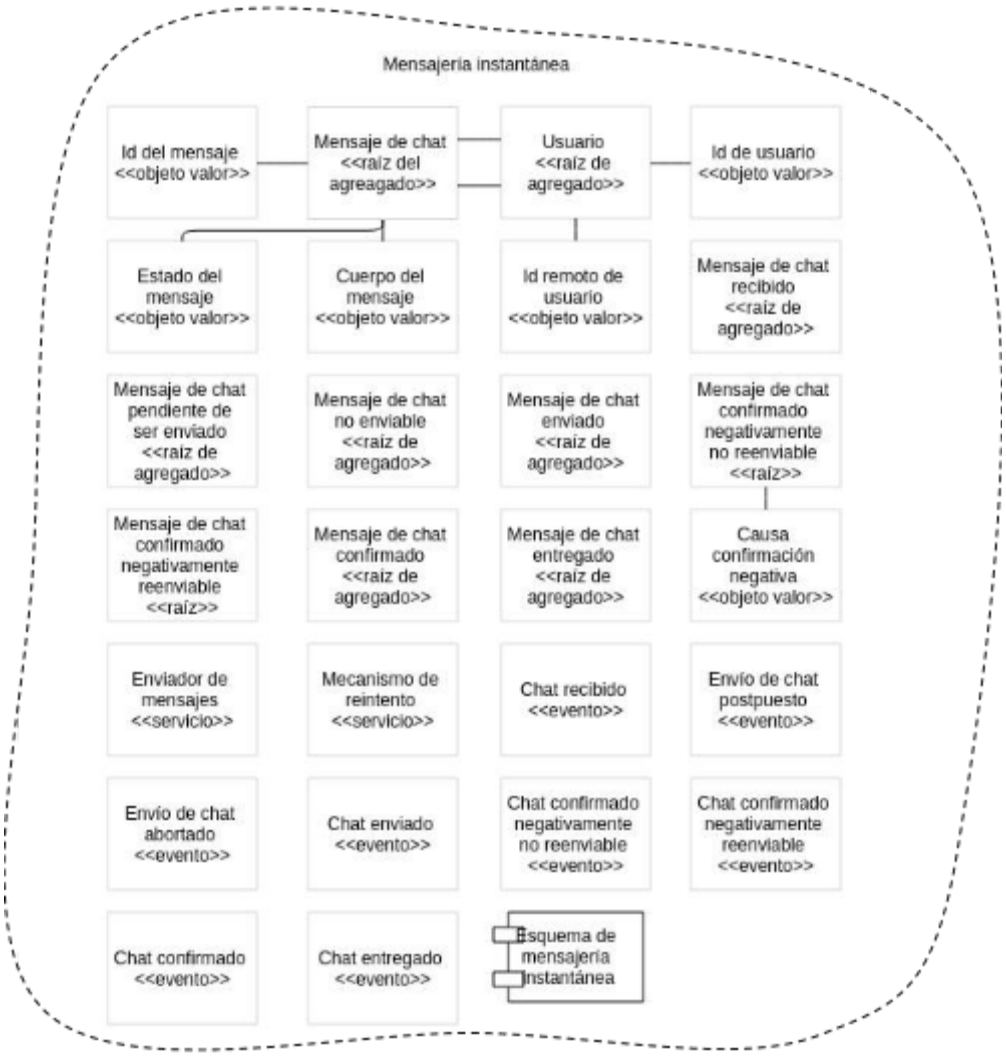
- 10 agregados: Mensaje de chat, Usuario, Mensaje de chat recibido, Mensaje de chat pendiente de ser enviado, Mensaje de chat no enviable, Mensaje de chat enviado, Mensaje de chat confirmado negativamente reenviable, Mensaje de chat confirmado negativamente no reenviable, Mensaje de chat confirmado, Mensaje de chat entregado.

7.3.3.2. El modelo

Lo que falta por añadir al modelo:

- 2 servicios: el enviador de mensajes y el mecanismo de reintento.
- 8 eventos: chat recibido, envío de chat postpuesto, envío de chat abortado, chat enviado, chat confirmado negativamente no reenviable, chat confirmado negativamente reenviable, chat confirmado, chat entregado.

El modelo definitivo:


Mensajería instantánea
Id del mensaje
<<objeto valor>>
Mensaje de chat
<<raíz del
agreagado>>
Usuario
<<raíz de
agregado>>
Id de usuario
<<objeto valor>>
Estado del
mensaje
<<objeto valor>>
Cuerpo del
mensaje
<<objeto valor>>
Id remoto de
usuario
<<objeto valor>>
Mensaje de chat
recibido
<<raíz de
agregado>>
Mensaje de chat
pendiente de
ser enviado
<<raíz de
agregado>>
Mensaje de chat
no enviable
<<raíz de
agregado>>
Mensaje de chat
enviado
<<raíz de
agregado>>
Mensaje de chat
confirmado
negativamente
no reenviable
<<raíz>>
Mensaje de chat
confirmado
negativamente
reenviable
<<raíz>>
Mensaje de chat
confirmado
<<raíz de
agregado>>
Mensaje de chat
entregado
<<raíz de
agregado>>
Causa
confirmación
negativa
<<objeto valor>>
Enviador de
mensajes
<<servicio>>
Mecanismo de
reintento
<<servicio>>
Chat recibido
<<evento>>
Envío de chat
postpuesto
<<evento>>
Envío de chat
abortado
<<evento>>
Chat enviado
<<evento>>
Chat confirmado
negativamente
no reenviable
<<evento>>
Chat confirmado
negativamente
reenviable
<<evento>>
Chat confirmado
<<evento>>
Chat entregado
<<evento>>
Esquema de
mensajería
instantánea

7.3.4. Geodirectorio de profesionales

7.3.4.1. Los agregados

Determinaremos los agregados a partir de las siguientes funcionalidades:

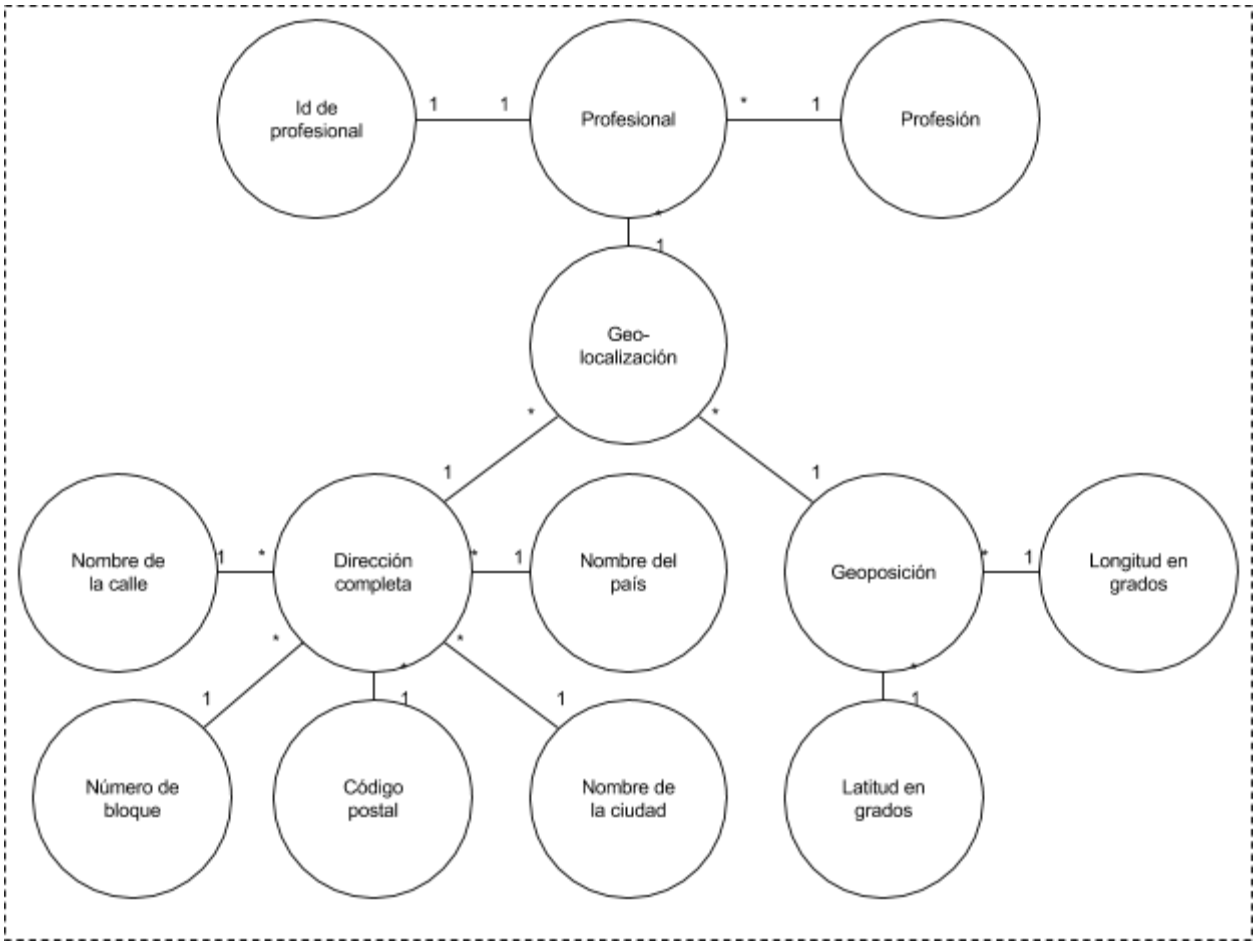
- Permitir al profesional especificar sus datos de ubicación.
- Permitir al profesional actualizar sus datos de ubicación.
- Permitir al profesional especificar su profesión.
- Permitir al cliente obtener un informe con profesionales cercanos.
- Permitir al cliente obtener un informe con profesionales cercanos de determinada profesión.

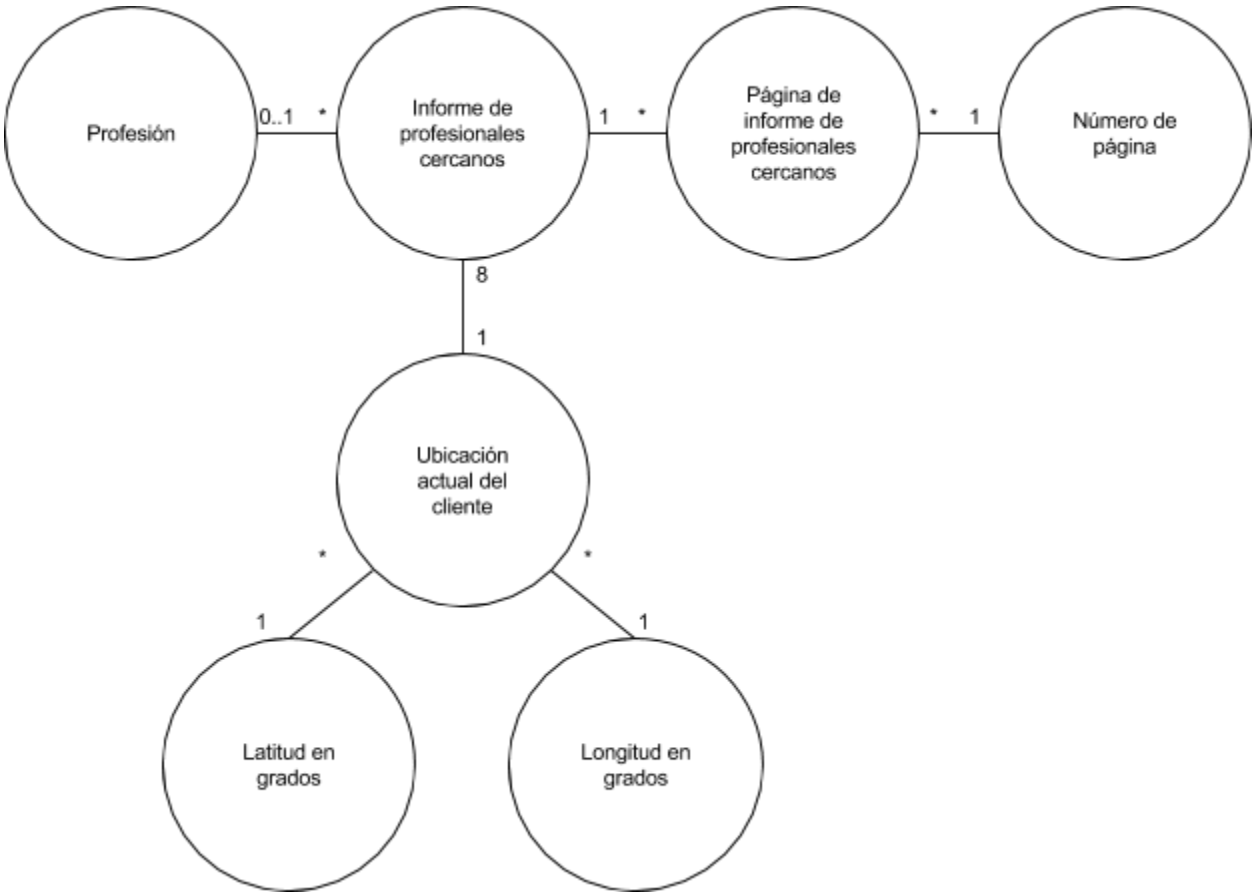
Y de las siguientes reglas de negocio:

- Los datos de ubicación de un profesional consisten en su dirección completa y en su geoposición.
- La dirección completa de un profesional consiste en el nombre de la calle, el número de bloque, el código postal, el nombre de la ciudad y el nombre del país.
- La geoposición de un profesional la determinan su latitud y su longitud, ambas expresadas en grados.
- Las profesiones que contemplamos por el momento son siete: carpintero, enyesador, techador, pintor, albañil, electricista y fontanero.

- Los informes de profesionales cercanos se realizan con respecto a la ubicación actual del cliente.
- Los informes de profesionales cercanos se encuentran paginados, y estas páginas están numeradas.

De todo ello obtenemos lo siguiente:





De lo que obtenemos:

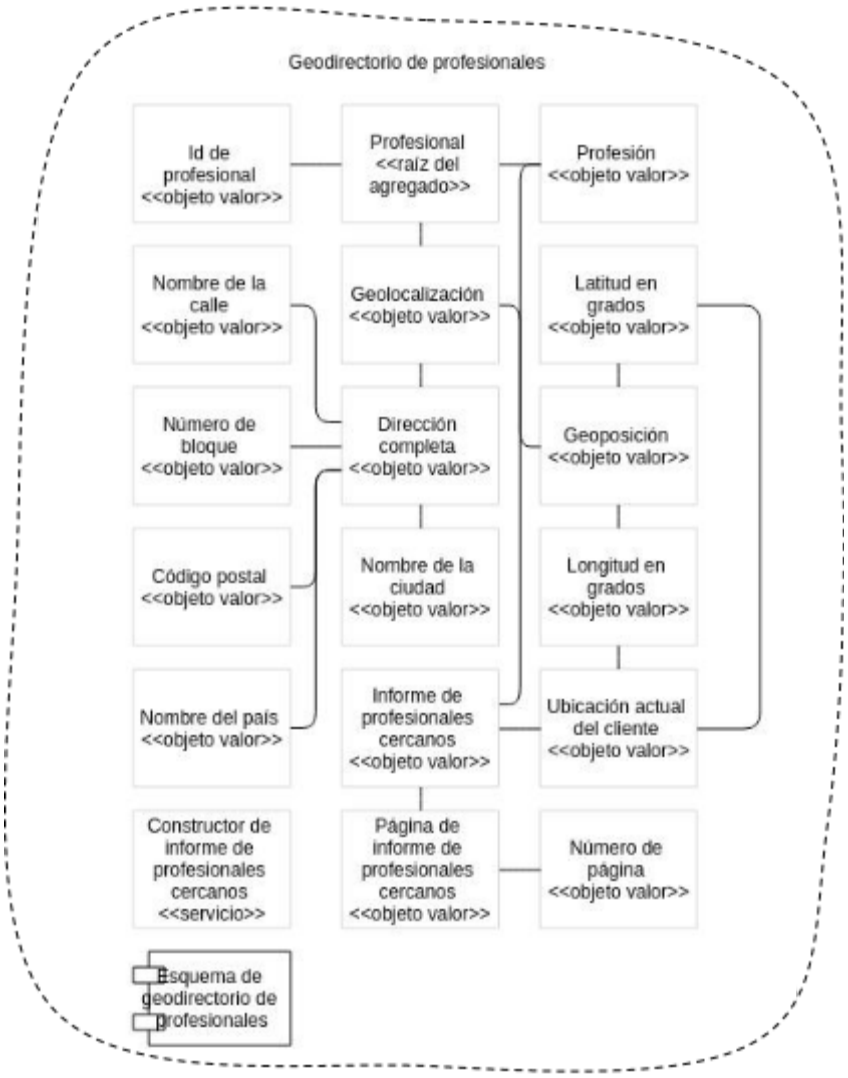
- 1 agregado en la parte del modelo de escritura: Profesional.
- 1 objeto valor, que incluye otros objetos valor, en la parte del modelo de lectura: Informe de profesionales cercanos.

7.3.4.2. El modelo

Lo que falta por añadir al modelo:

- 1 servicio: constructor de informes de profesionales cercanos.

El modelo definitivo:



7.3.5. Pagos de tareas

7.3.5.1. Los agregados

Determinaremos los agregados a partir de las siguientes funcionalidades:

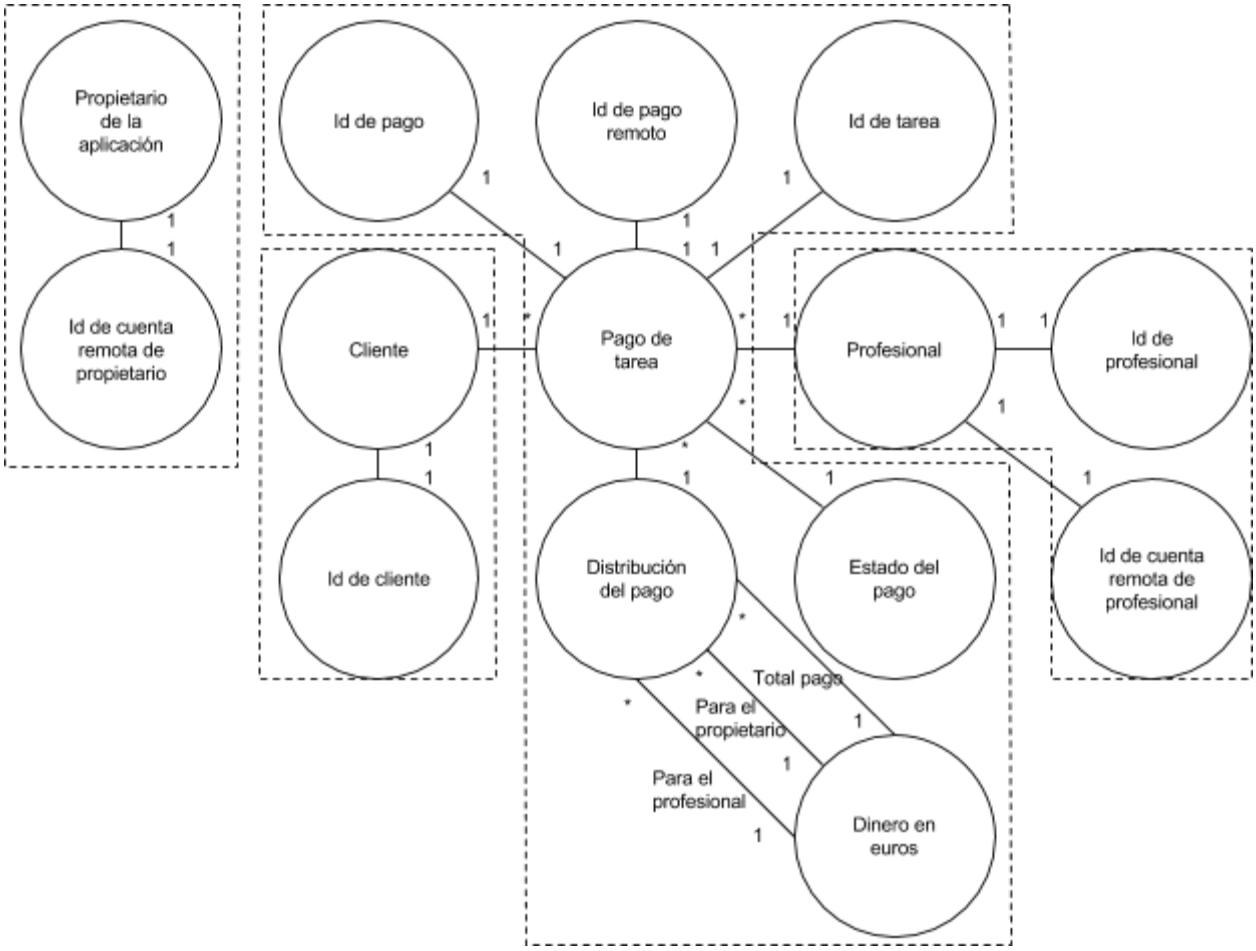
- Permitir al profesional especificar la cuenta en la que desea recibir sus pagos.
- Permitir al cliente obtener un informe con el total cobrado por un profesional a través de la aplicación.
- Permitir al cliente pagar al profesional que le ha realizado una tarea.
- Recompensar a aquellos usuarios cuyas invitaciones han sido utilizadas por otros usuarios para registrarse. La recompensa consistirá en una participación en la tasa que cobre el sistema a los pagos realizados o percibidos por el usuario invitado.

Y de las siguientes reglas de negocio:

- La especificación de la cuenta por parte del profesional consiste en que este facilite el identificador de una cuenta registrada en el proveedor de servicios de pago.
- Solamente se pueden pagar las tareas que han sido realizadas.
- El pago se distribuye de la siguiente manera: 1% para el propietario de la aplicación, 99% para el profesional.
- El encargado de pagar las tasas del proveedor de servicios de pago es el profesional.

- Por razones de seguridad, no se ingresará el dinero en la cuenta del profesional hasta pasados diez días desde el momento del pago.

De todo ello obtenemos lo siguiente:



De lo que obtenemos:

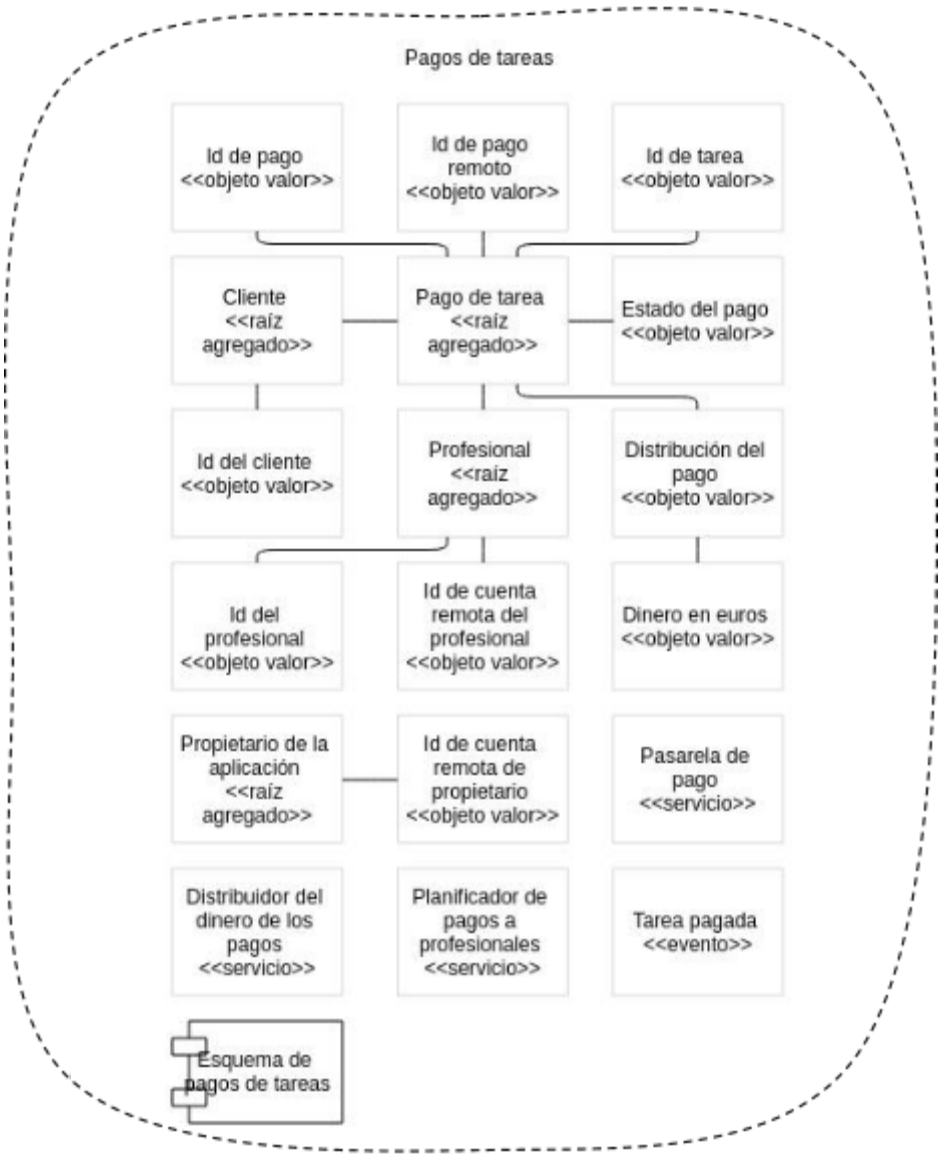
- 4 agregados: Pago de tarea, Cliente, Profesional y Propietario.

7.3.5.2. El modelo

Lo que falta por añadir al modelo:

- 3 servicios: pasarela de pago, distribuidor del dinero de los pagos, planificador de pagos a profesionales.
- 1 evento: tarea pagada.

El modelo definitivo:


Pagos de tareas
Id de pago
<<objeto valor>>
Id de pago remoto
<<objeto valor>>
Id de tarea
<<objeto valor>>
Cliente
<<raíz agregado>>
Pago de tarea
<<raíz agregado>>
Estado del pago
<<objeto valor>>
Id del cliente
<<objeto valor>>
Profesional
<<raíz agregado>>
Distribución del pago
<<objeto valor>>
Id del profesional
<<objeto valor>>
Id de cuenta remota del profesional
<<objeto valor>>
Dinero en euros
<<objeto valor>>
Propietario de la aplicación
<<raíz agregado>>
Id de cuenta remota de propietario
<<objeto valor>>
Pasarela de pago
<<servicio>>
Distribuidor del dinero de los pagos
<<servicio>>
Planificador de pagos a profesionales
<<servicio>>
Tarea pagada
<<evento>>
Esquema de pagos de tareas

7.3.6. Tareas

7.3.6.1. Los agregados

Determinaremos los agregados a partir de las siguientes funcionalidades:

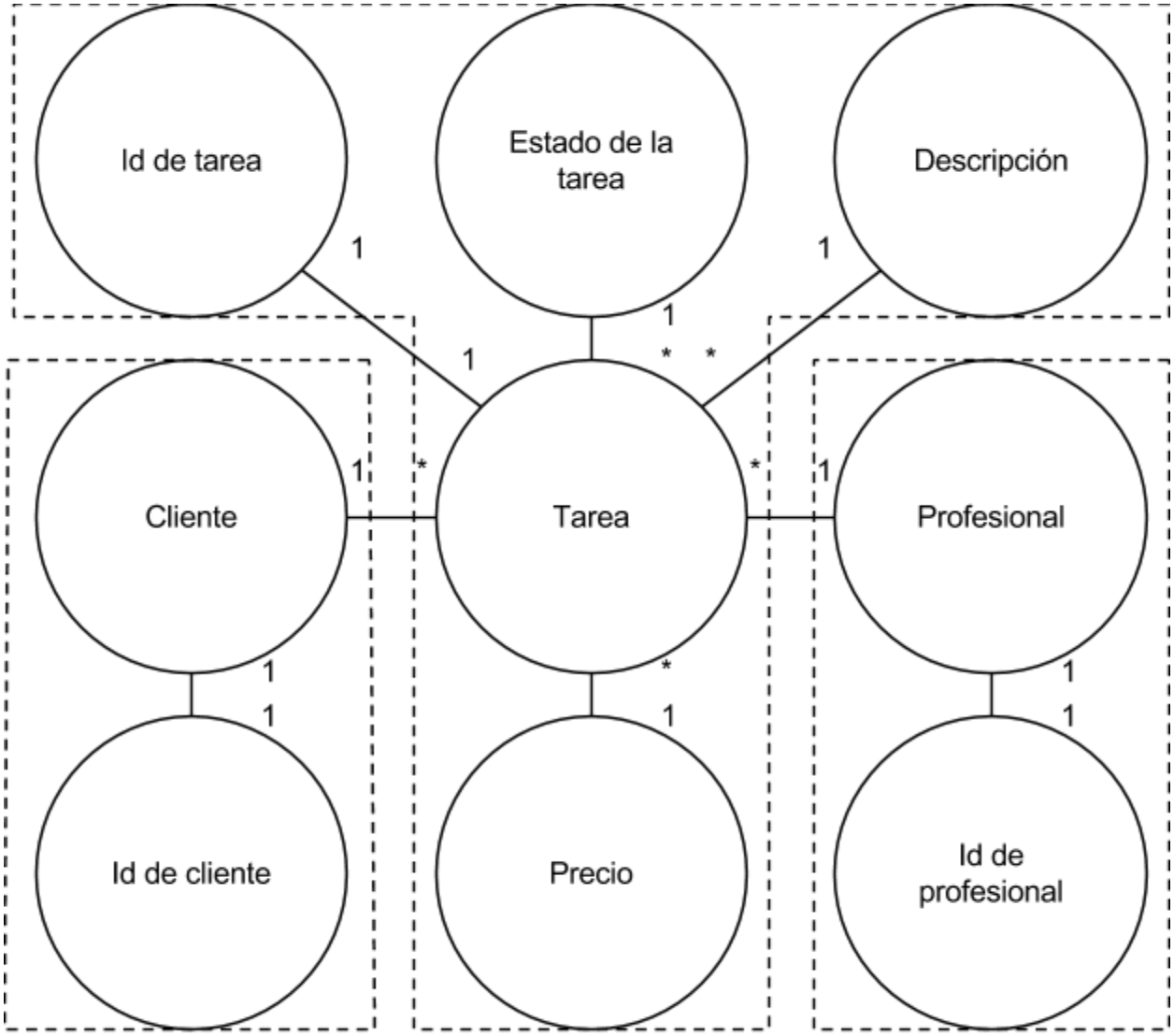
- Permitir a los clientes solicitar una tarea a un profesional.
- Permitir a los profesionales realizar un presupuesto para una tarea solicitada por un cliente.
- Permitir a los clientes rechazar un presupuesto realizado por un profesional.
- Permitir a los clientes aceptar un presupuesto realizado por un profesional.
- Permitir a los profesionales indicar que ya han realizado la tarea que les encomendó el cliente.
- Permitir al cliente obtener un informe con las tareas que ha solicitado.
- Permitir al profesional obtener un informe con las tareas que se le han solicitado.
- Permitir al cliente obtener un informe con las tareas que ha solicitado a determinado profesional.
- Permitir al profesional obtener un informe con las tareas que le han sido solicitadas por determinado cliente.

Y de las siguientes reglas de negocio:

- Una solicitud de tarea incluye la descripción de la tarea.

- Un presupuesto de tarea incluye el precio presupuestado.
- Se presupuestan tareas solicitadas únicamente.
- Solamente una tarea cuyo presupuesto ha sido aceptado puede marcarse como realizada.
- Solamente las tareas realizadas pueden ser pagadas.
- Solamente las tareas pagadas pueden ser valoradas.
- Una tarea debe estar en cualquiera de los siguientes siete estados: solicitada, presupuestada, presupuesto rechazado, presupuesto aceptado, realizada, pagada y valorada.

De todo ello obtenemos lo siguiente:



De lo que obtenemos:

- 3 agregados: Tarea, Cliente y Profesional.

7.3.6.2. El modelo

Lo que falta por añadir al modelo:

- Nada.

El modelo definitivo:

7.3.7. Valoraciones de tareas

7.3.7.1. Los agregados

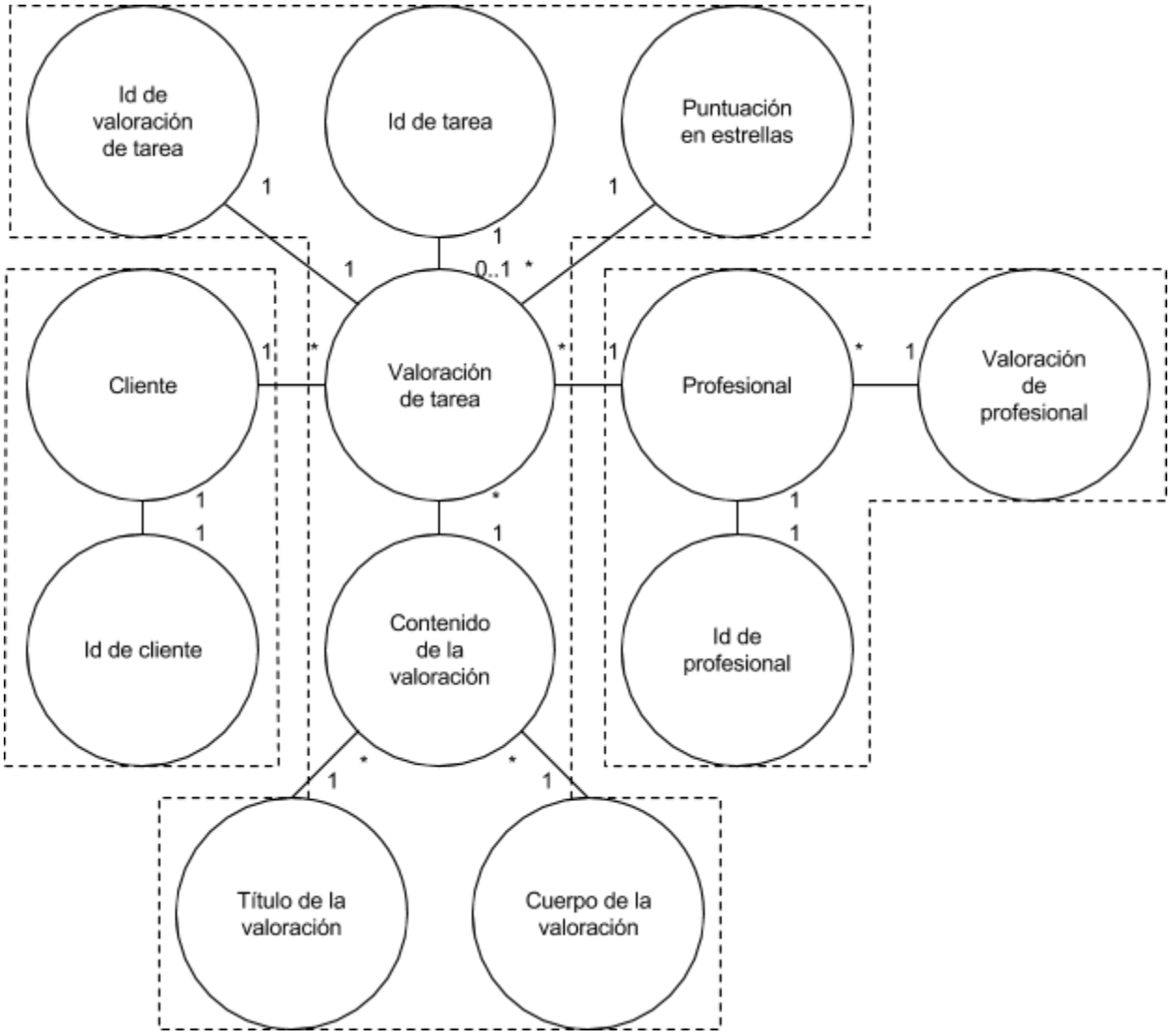
Determinaremos los agregados a partir de las siguientes funcionalidades:

- Permitir al cliente valorar el desempeño del profesional en la realización de determinada tarea.
- Permitir al cliente actualizar la valoración dada al trabajo realizado por un profesional.
- Permitir al cliente obtener un informe con el historial de valoraciones de un profesional.
- Permitir al cliente obtener un informe con la valoración media de un profesional.
- Permitir al profesional obtener un informe con su historial de valoraciones recibidas.
- Permitir al profesional obtener un informe con su valoración media.

Y de las siguientes reglas de negocio:

- Solamente se pueden valorar aquellas tareas que hayan sido pagadas.
- La valoración incluye una puntuación en estrellas y un contenido.
- La puntuación puede ser de 1, 2, 3, 4 o 5 estrellas.
- El contenido consiste en un título y un cuerpo.
- La valoración media de un profesional es la media de las estrellas obtenidas en las tareas realizadas ponderadas las estrellas por el precio de la tarea.

De todo ello obtenemos lo siguiente:



De lo que obtenemos:

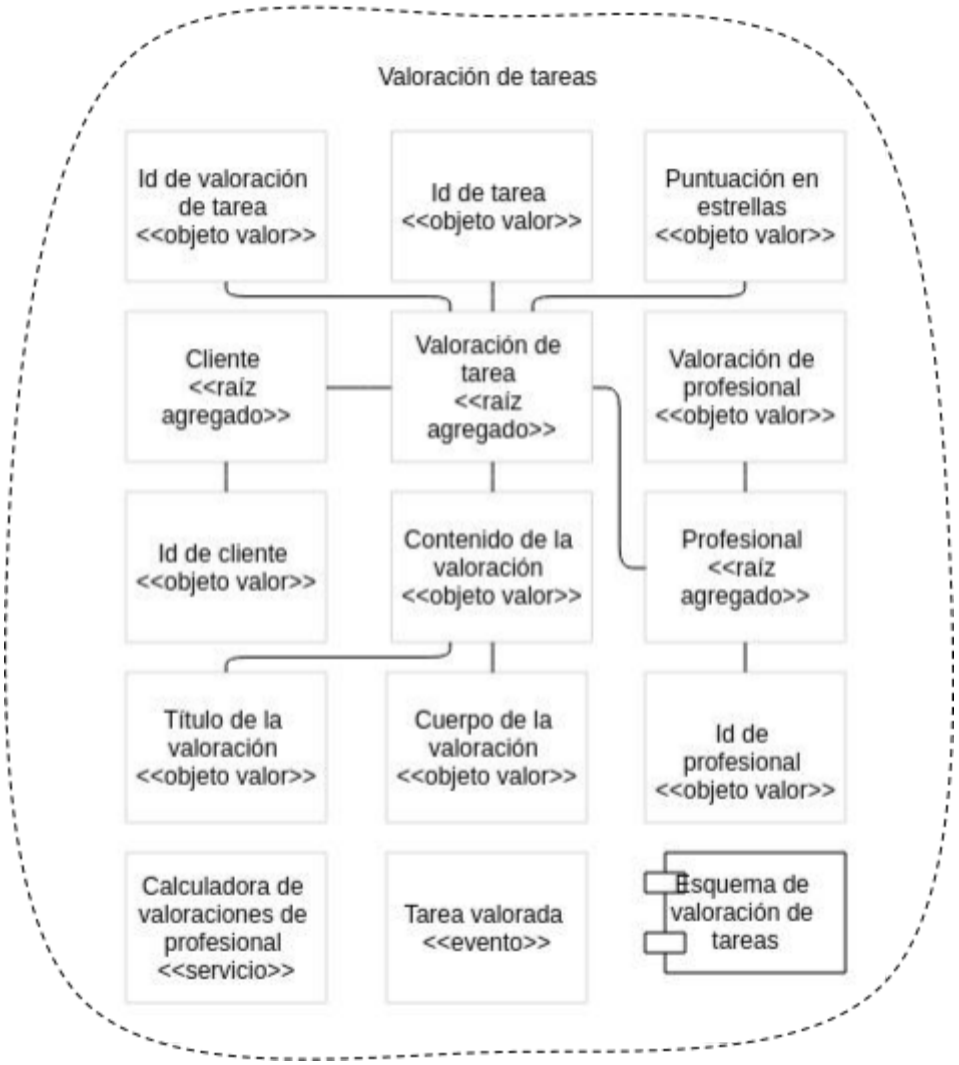
- 3 agregados: Valoración de tarea, Cliente y Profesional.

7.3.7.2. El modelo

Lo que falta por añadir al modelo:

- 1 servicio: calculadora de valoraciones de profesional.
- 1 evento: tarea valorada.

El modelo definitivo:



7.4. Arquitectura del sistema

El estilo arquitectónico aplicado a los siete componentes del sistema es una arquitectura en capas con las dependencias invertidas.

Por un lado, las capas de nuestros componentes son cuatro:

En primer lugar, la interfaz de usuario o API, que lo que hace es publicar los servicios de la capa de servicios de aplicación para que sean accesibles vía HTTP. Estas APIs están publicadas haciendo uso de controladores, que son los elementos principales de esta capa.

En segundo lugar, la capa de servicios de aplicación que contiene todos los casos de uso del componente. Se encarga de orquestar el dominio para satisfacer estos casos de uso, y es por este motivo que es la responsable de gestionar las transacciones. También contiene lógica de la aplicación que no pertenece al dominio, como por ejemplo el envío de mensajes de correo electrónico a un administrador.

En tercer lugar, la capa del dominio contiene el modelo del componente. Debería incluir todas las reglas de negocio, y velar por el cumplimiento de todos los invariantes del dominio.

En cuarto lugar, la capa de infraestructura, que se encarga de aislar todas las dependencias tecnológicas para que no trasciendan al resto del código.

Por otro lado, al estar las dependencias invertidas nadie depende de la capa de infraestructura, que se dedica a implementar interfaces del resto de capas. Por lo tanto, tenemos que la capa de interfaz de usuario depende de la capa de servicios de aplicación, que la capa de servicios

de aplicación depende de la capa del dominio, que la capa del dominio no depende de nadie, y que la capa de infraestructura depende de las otras tres.

El uso que se ha hecho de la arquitectura en capas es bastante estricto. Esto quiere decir que, en general, la capa de interfaz de usuario no accede directamente a elementos de la capa de dominio.

Hay otros dos estilos arquitectónicos que se han aplicado parcialmente sobre sendos componentes:

Por un lado, en el componente de mensajería instantánea se ha aplicado prácticamente una arquitectura de puertos y adaptadores. En efecto, tenemos dos puertos de entrada/salida, uno para HTTP y otro para XMPP, y luego los adaptadores de entrada y de salida, para adaptar la entrada que nos llega por el puerto a la interfaz de los servicios de aplicación, y para adaptar la entrada que nos llega de los servicios de aplicación al formato del puerto, respectivamente. El paso que haría falta para migrar de la actual arquitectura en capas a esta es mover la actual capa de interfaz de usuario a infraestructura, y organizar dicha capa de infraestructura en módulos correspondientes a puertos y adaptadores.

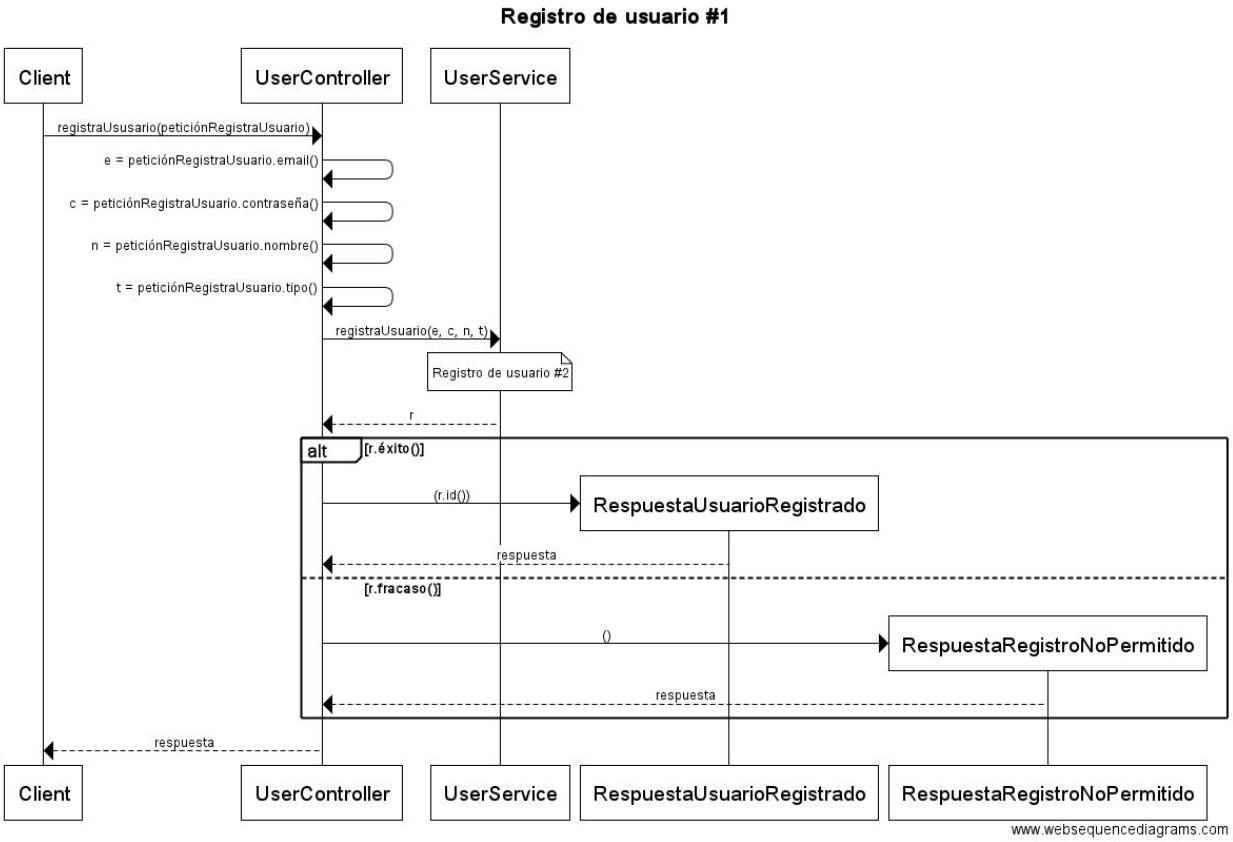
Por otro lado, en el componente de geodirectorio de profesionales se ha aplicado segregación de responsabilidades en comandos y consultas o Command Query Responsibility Segregation (CQRS), de modo que tenemos un modelo de lectura y otro de escritura, y en la parte de aplicación tenemos diferenciados los servicios de aplicación que atienden comandos de los que atienden consultas. Sin embargo, se sigue manteniendo en todo momento la arquitectura en capas, lo cual es posible porque ambos estilos arquitectónicos son combinables y parcialmente compatibles.

7.5. Diseño del comportamiento: diagramas de secuencia

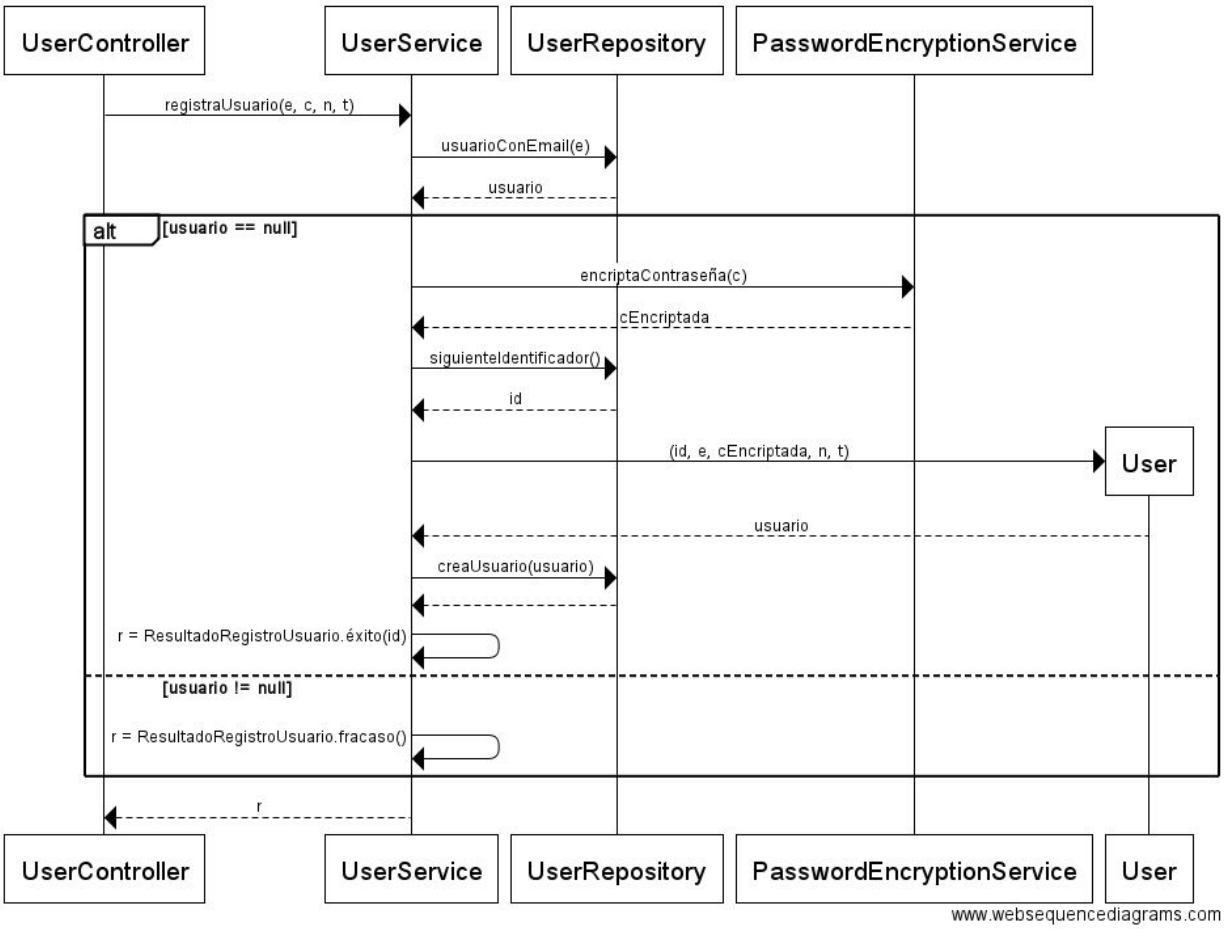
En esta sección mostraré los diagramas de secuencia más representativos de cada uno de los siete sistemas.

7.5.1. Diagramas de secuencia de identidad y acceso

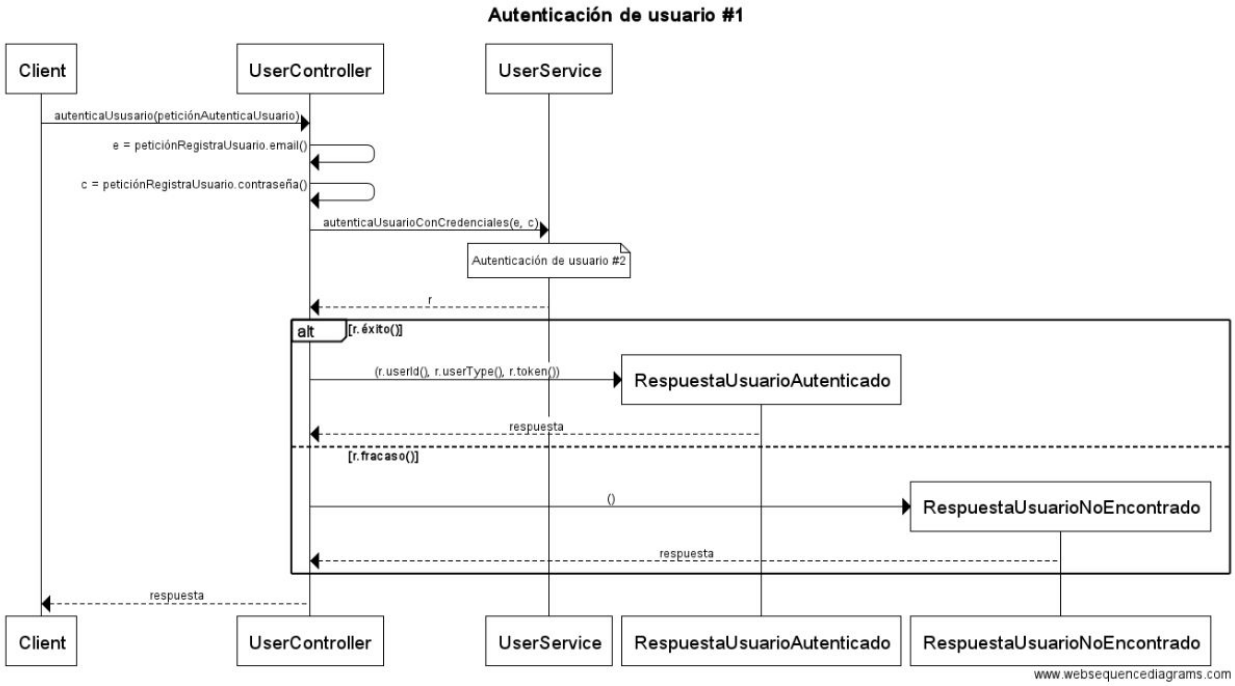
7.5.1.1. Registro de usuario

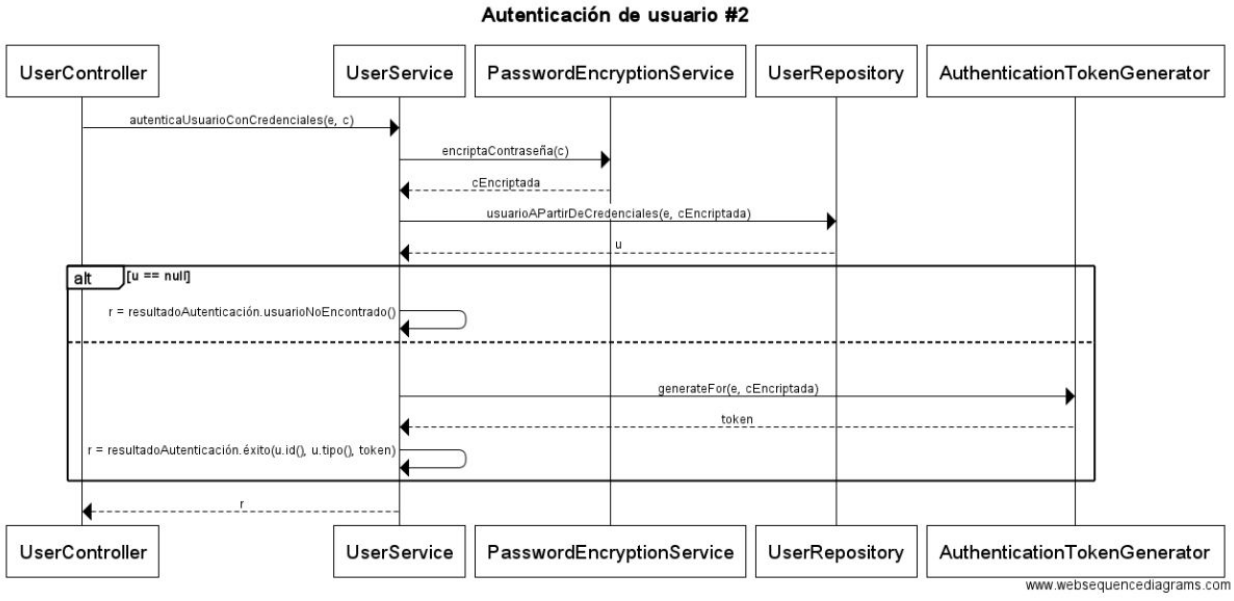


Registro de usuario #2



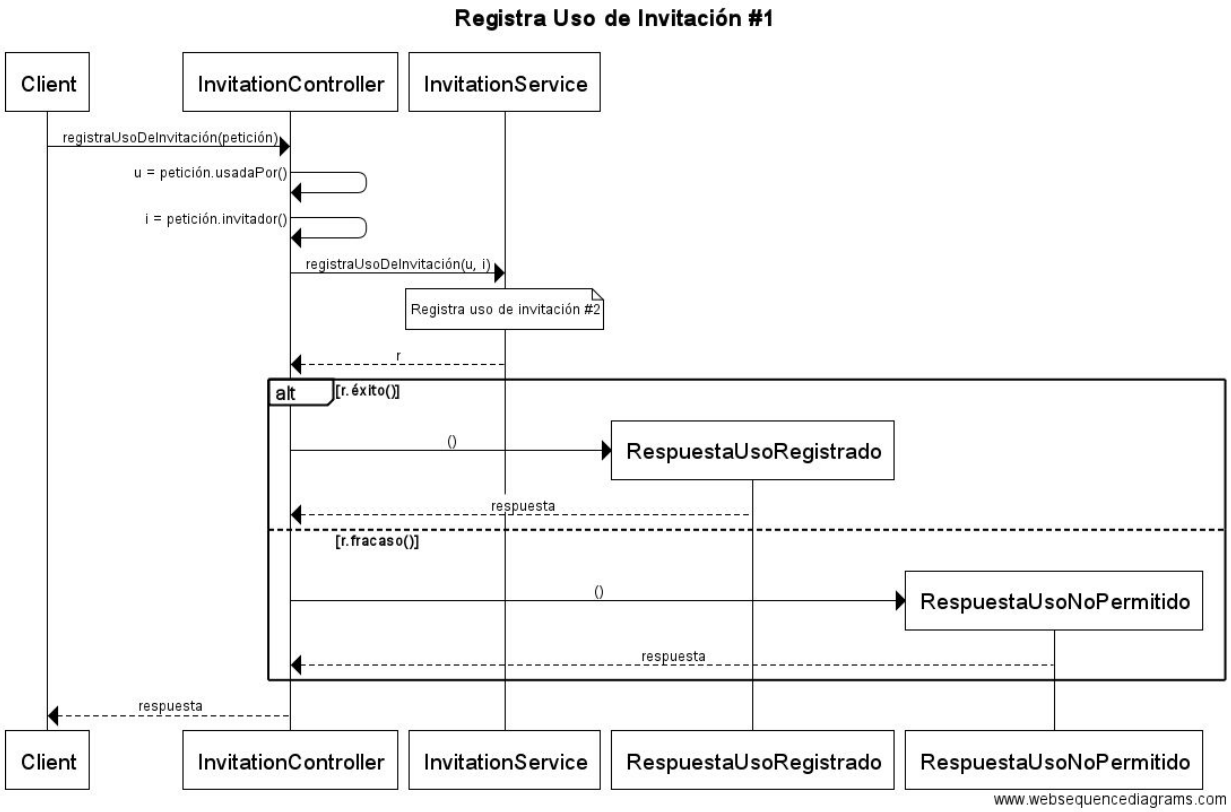
7.5.1.2. Autenticación de usuario



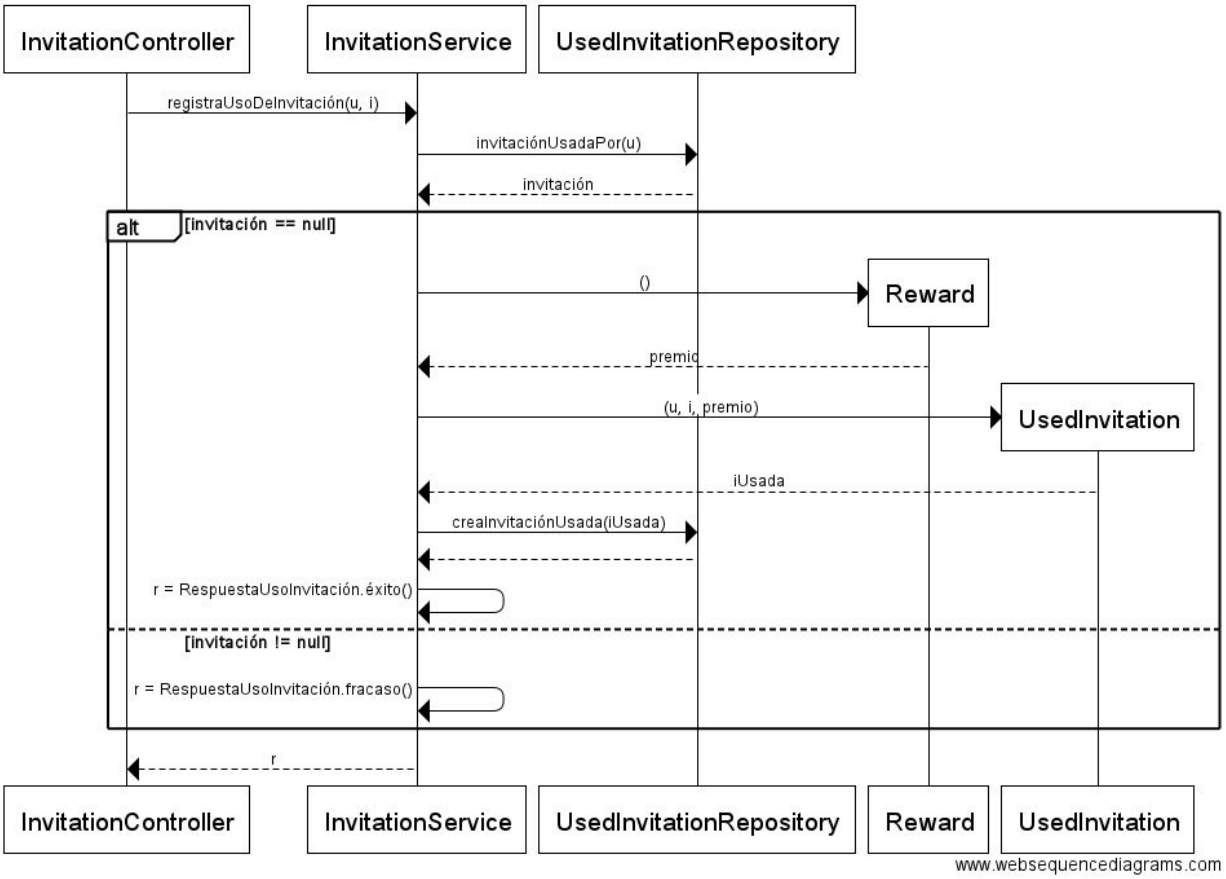


7.5.2. Diagramas de secuencia de invitaciones

7.5.2.1. Registro uso de invitación

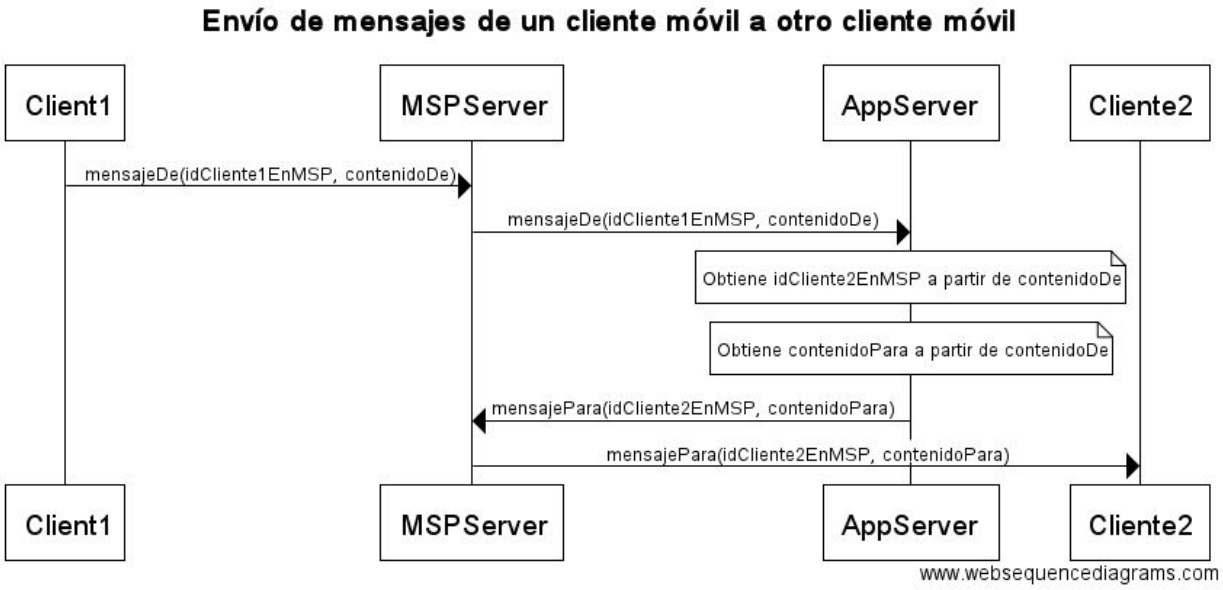


Registra Uso de Invitación #2



7.5.3. Diagramas de secuencia de mensajería instantánea

7.5.3.1. Diagrama de alto nivel de envío de mensajes de un cliente móvil a otro cliente móvil



Cabe aclarar que contenidoDe contiene dos cosas: el texto que se envía, y el identificador local (válido en nuestro servidor) del destinatario. A partir del identificador local del destinatario, nuestro servidor puede obtener el identificador de ese mismo destinatario en el servicio de mensajería externo, y enviar el mensaje para allá.

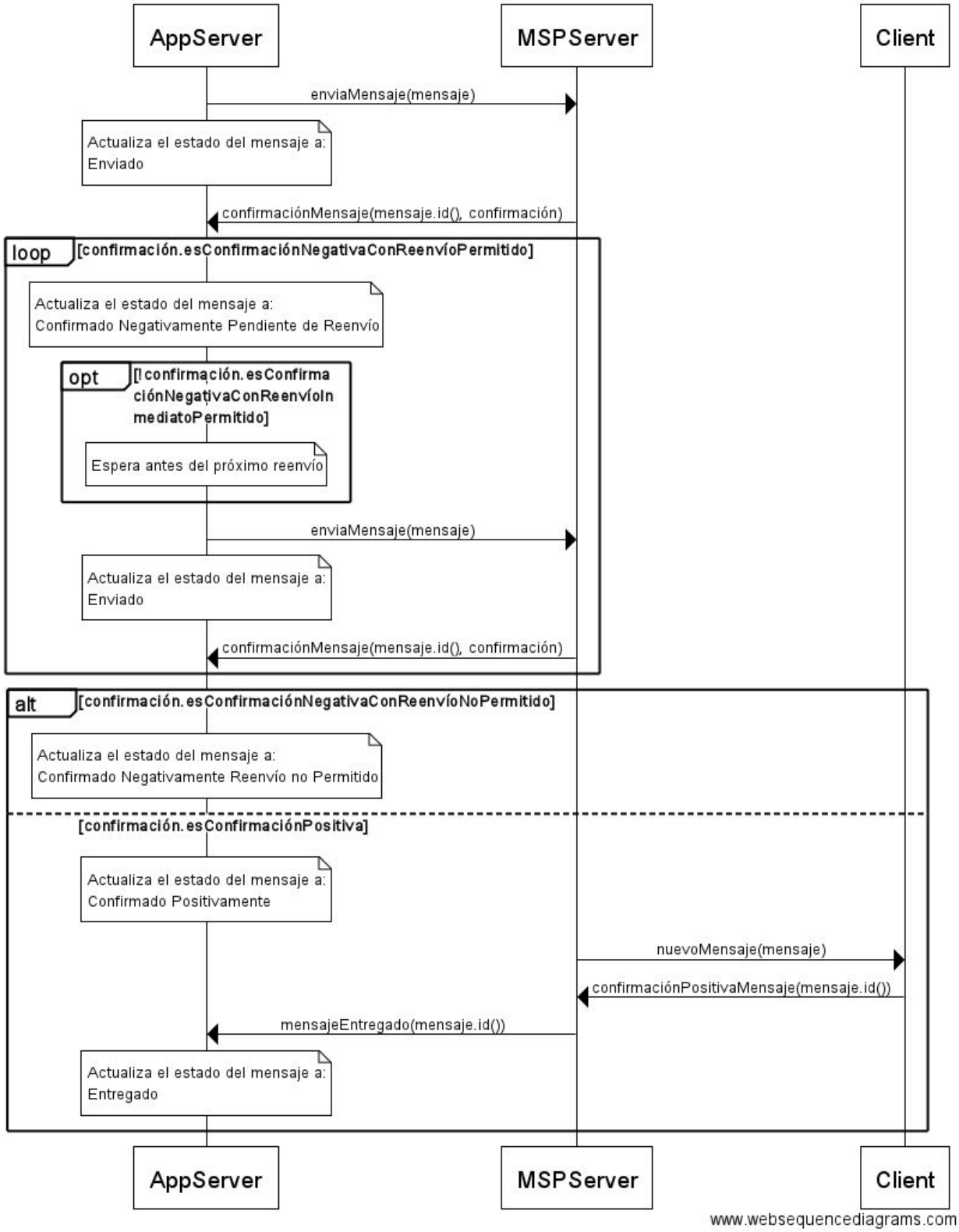
Por su parte, contenidoPara contiene también dos cosas: el texto que se envía, que lo obtendrá de contenidoDe, y el identificador local del remitente. Nuestro servidor obtendrá el identificador local del remitente a partir de su identificador remoto (válido en el servicio de mensajería externo).

Los clientes móviles trabajan con identificadores locales de usuarios. Envían al identificador local de un usuario, y reciben del identificador local de un usuario. El único identificador remoto, válido en el servicio de mensajería, que conocen es el suyo propio, pero no el de los demás. Por este motivo nuestra servidor transforma el mensaje como lo transforma antes de enviárselo al destinatario final.

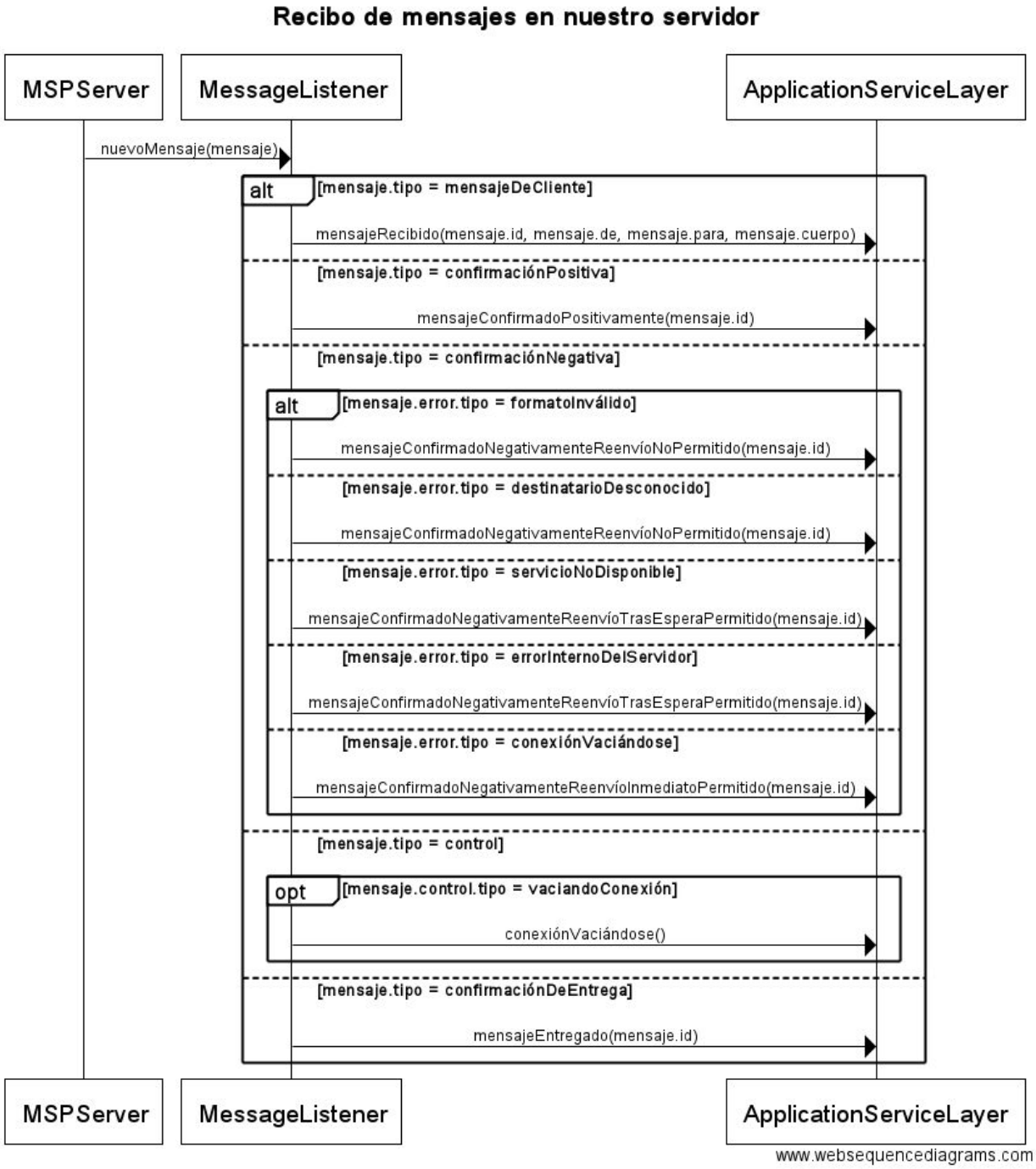
‘MSP’ es la abreviación de ‘Messaging Service Provider’.

7.5.3.2. Envío de mensajes de nuestro servidor al servidor del servicio de mensajería

Envío de mensajes de nuestro servidor al servidor del servicio de mensajería

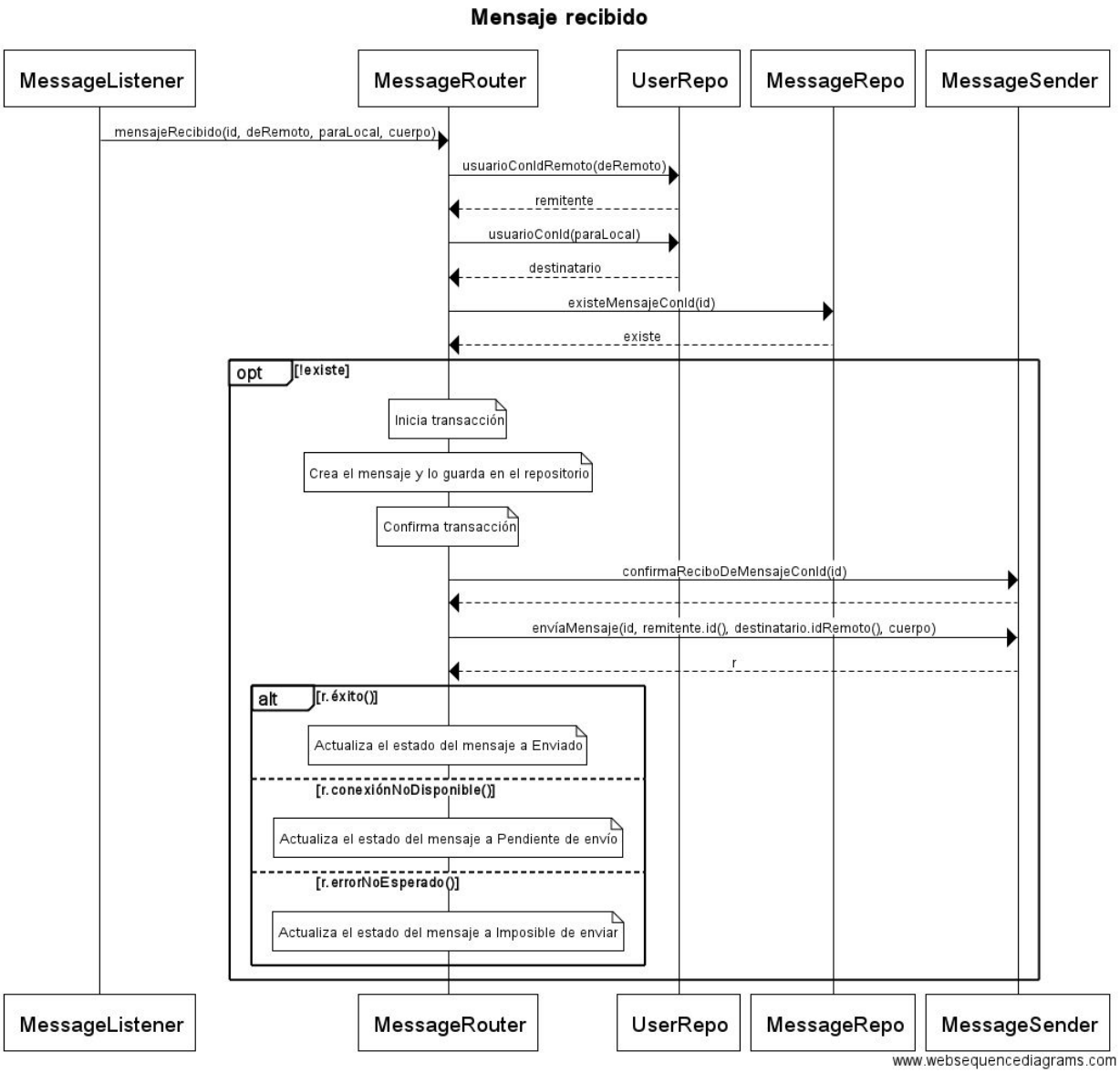


7.5.3.3. Recibo de mensajes en nuestro servidor



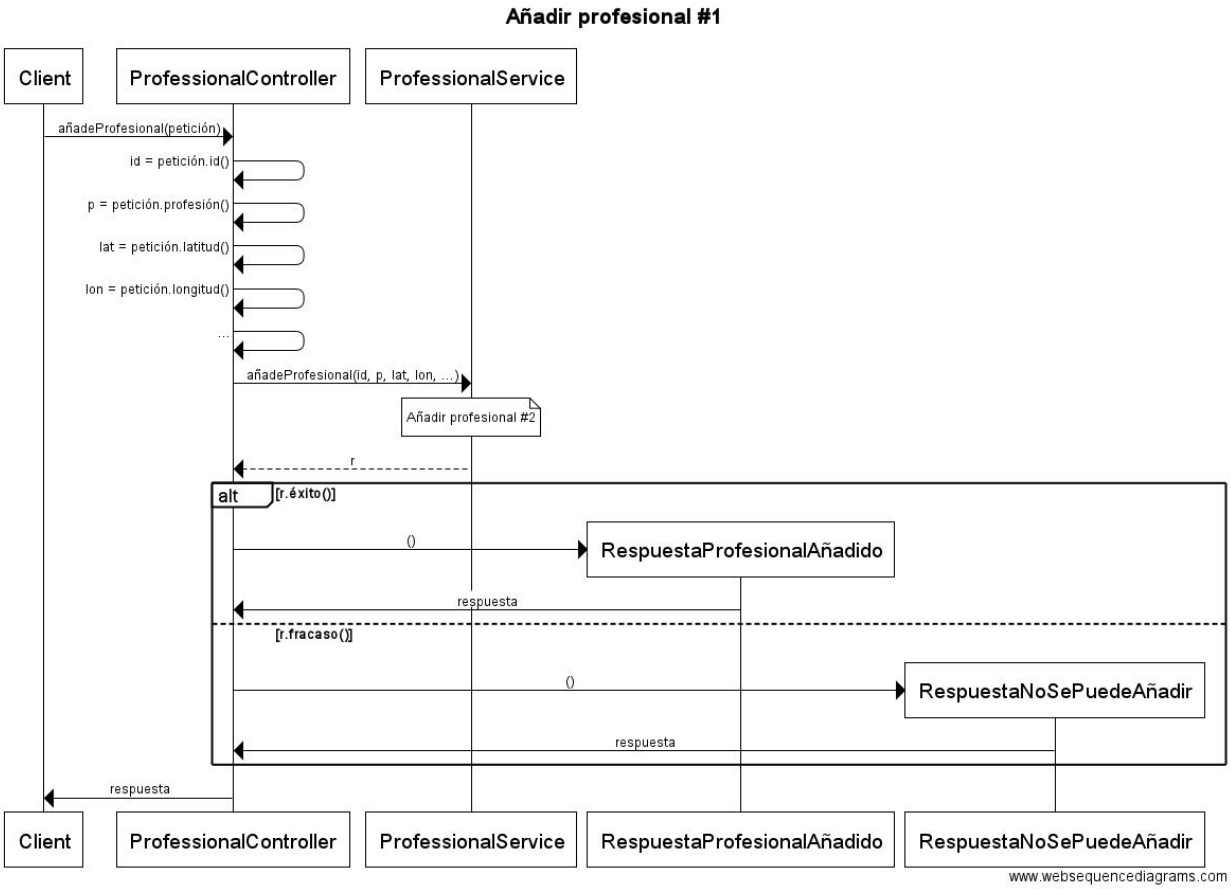
Utilizo el elemento del diagrama “ApplicationServiceLayer” para referirme a todos los servicios que habitan esa capa. Si me hubiera referido a los servicios individualmente, no me habrían cabido en el diagrama.

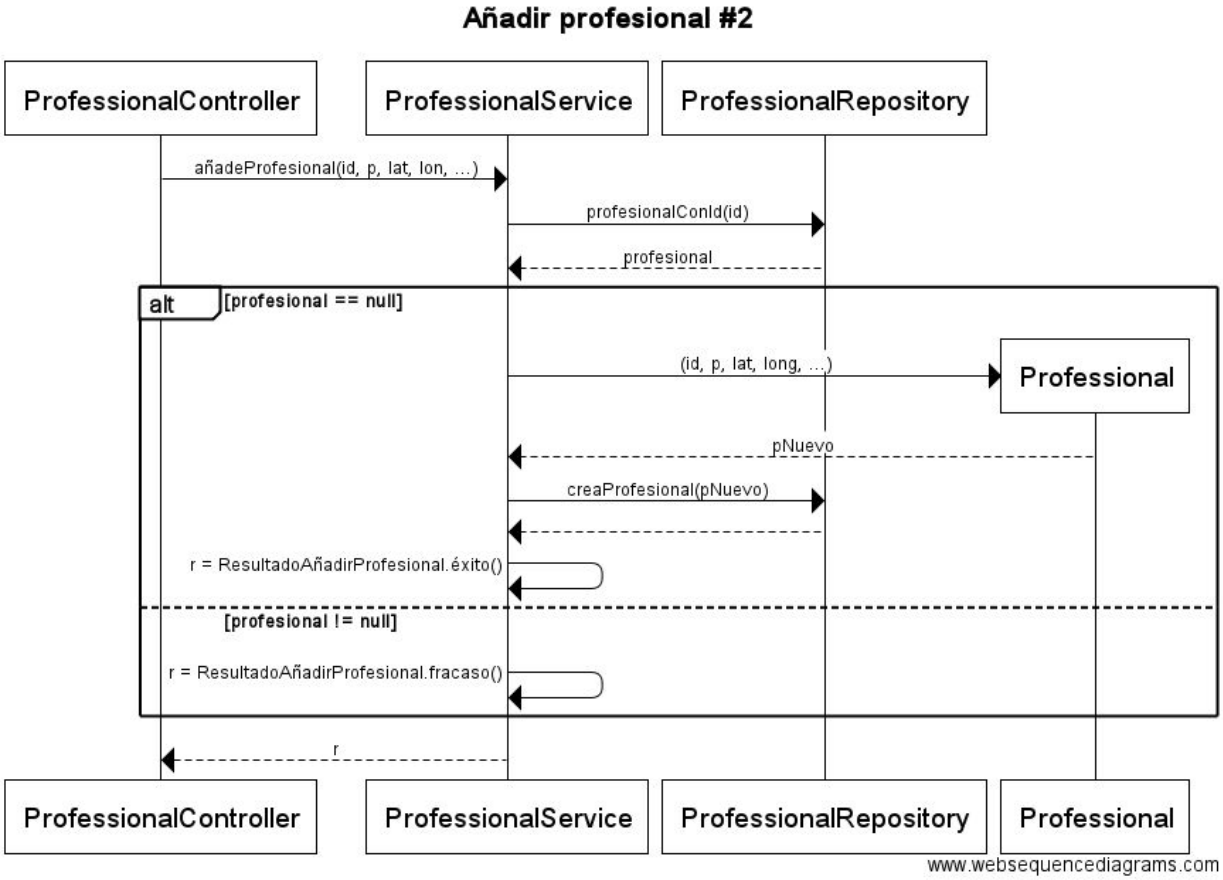
7.5.3.4. Mensaje recibido de usuario



7.5.4. Diagramas de secuencia de geodirectorio de profesionales

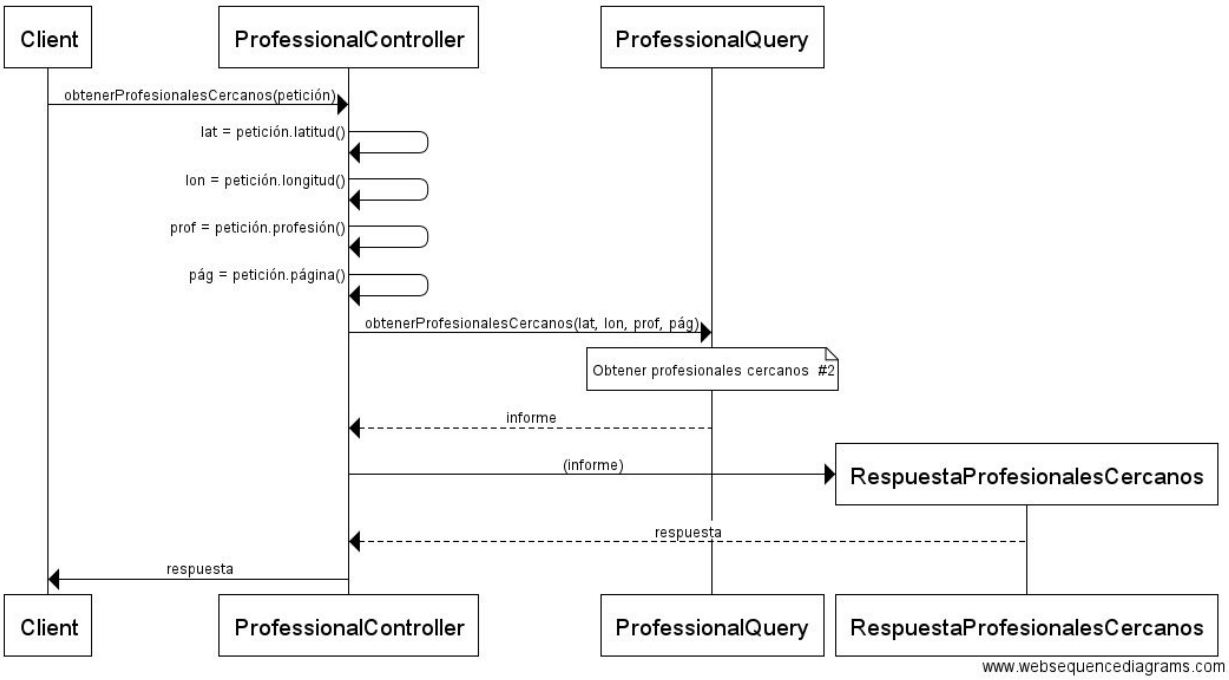
7.5.4.1. Añadir profesional



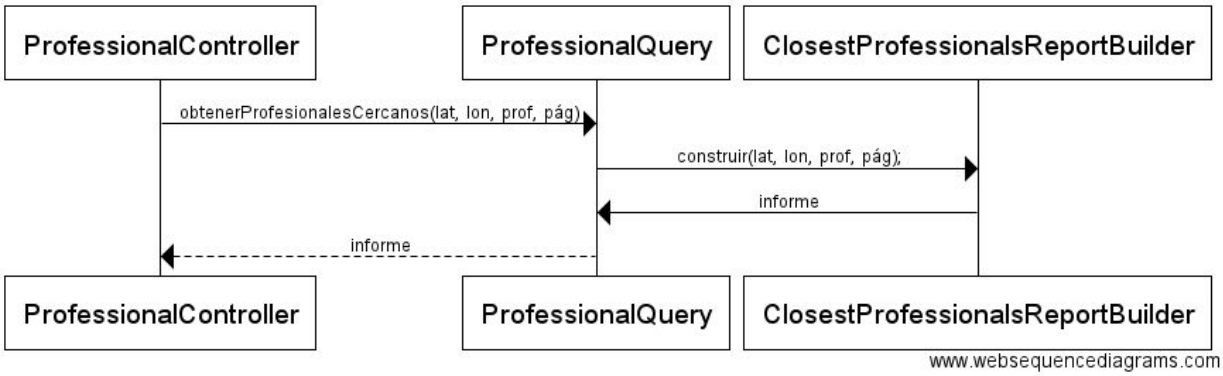


7.5.4.2. Consultar profesionales cercanos

Obtener profesionales cercanos #1

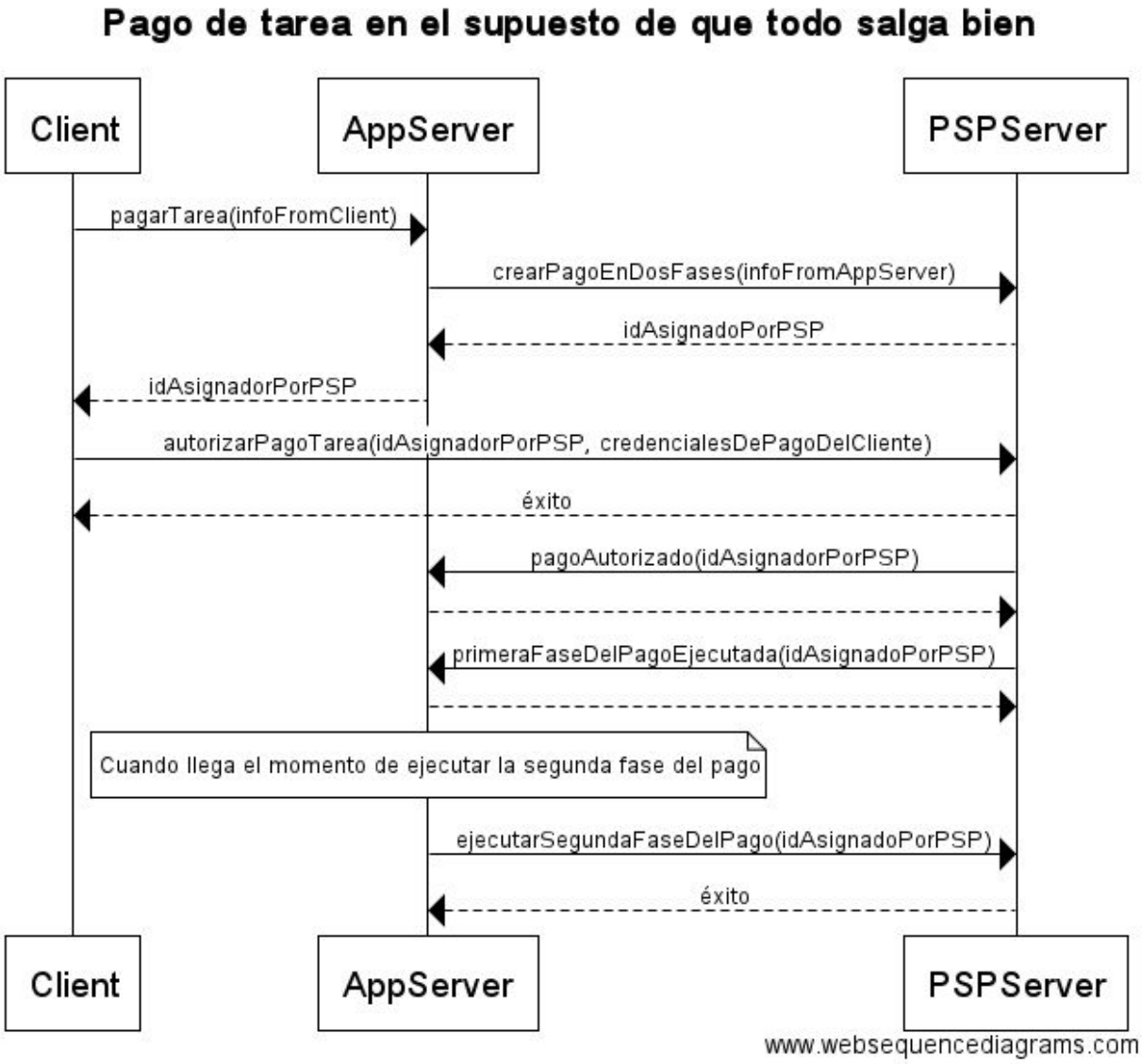


Obtener profesionales cercanos #2



7.5.5. Diagramas de secuencia de pagos de tareas

7.5.5.1. Diagrama de flujo general del pago de una tarea



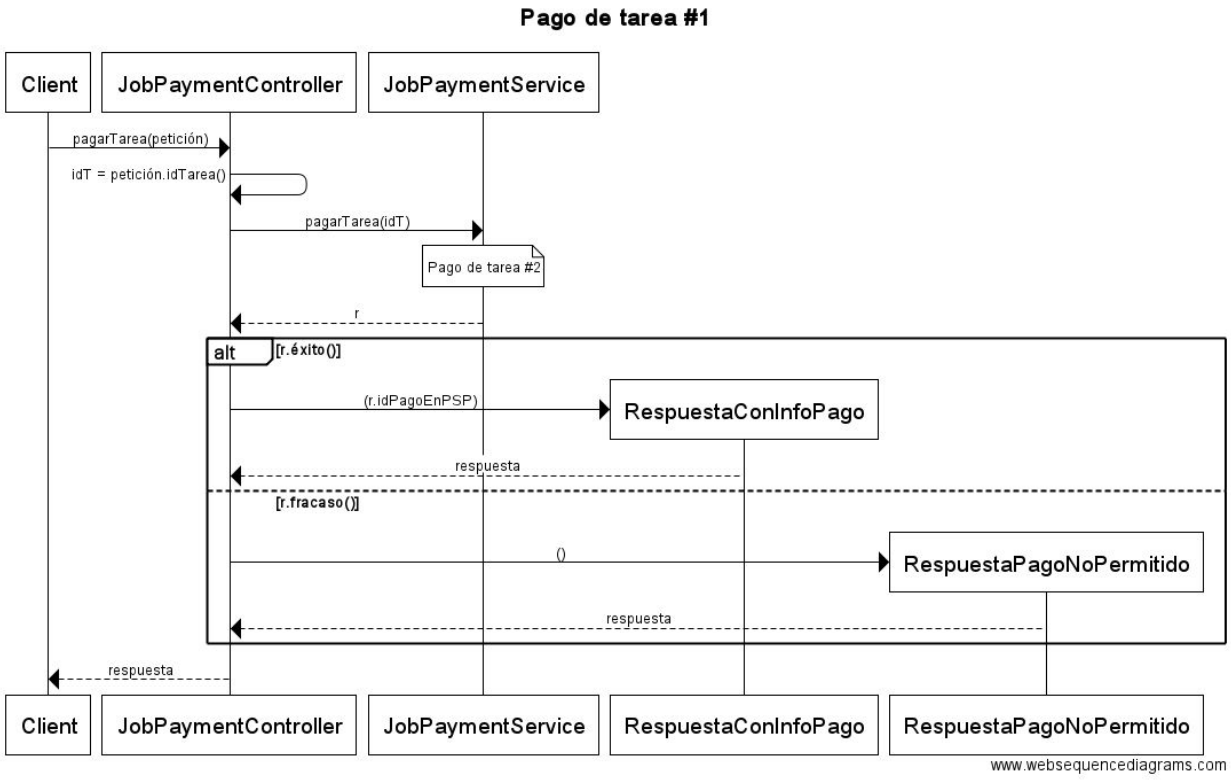
Los pagos de tareas se realizan en dos fases. En la primera fase, cobramos nosotros nuestra comisión como propietarios de la aplicación. Esta primera fase se ejecuta al autorizar el pago.

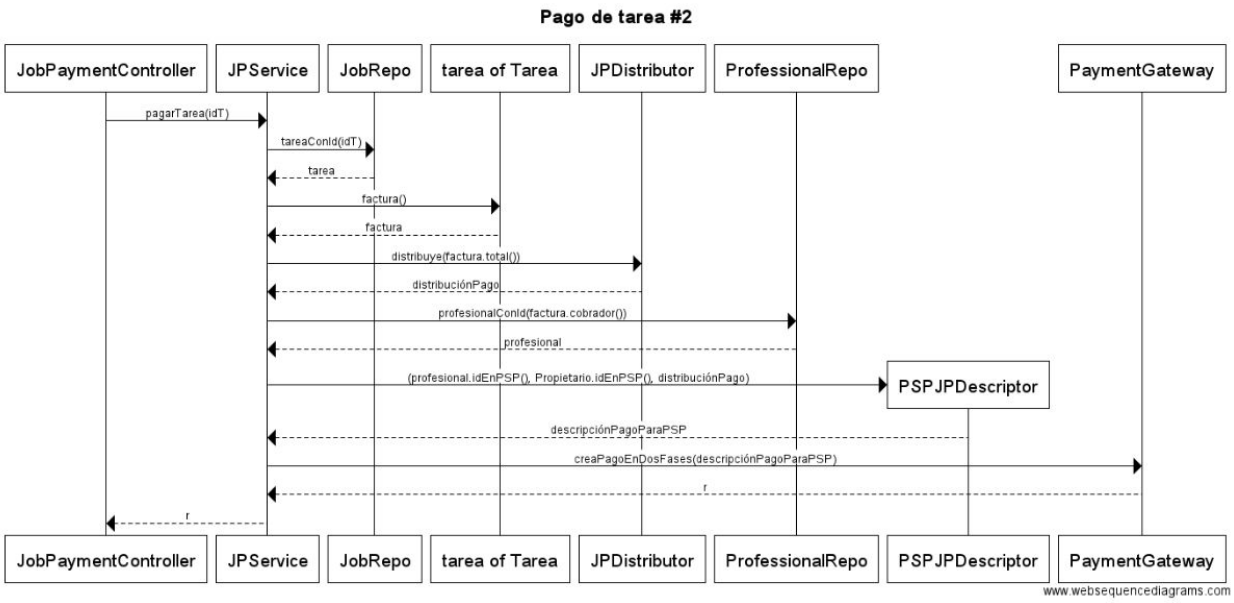
En la segunda fase, el que cobra es el profesional como realizador de la tarea. Damos orden para que se ejecute esta segunda fase diez días después de la autorización del pago.

Nótese que este diseño está fuertemente condicionado por el funcionamiento de las APIs de PayPal.

'PSP' es la abreviación de 'Payment Service Provider'.

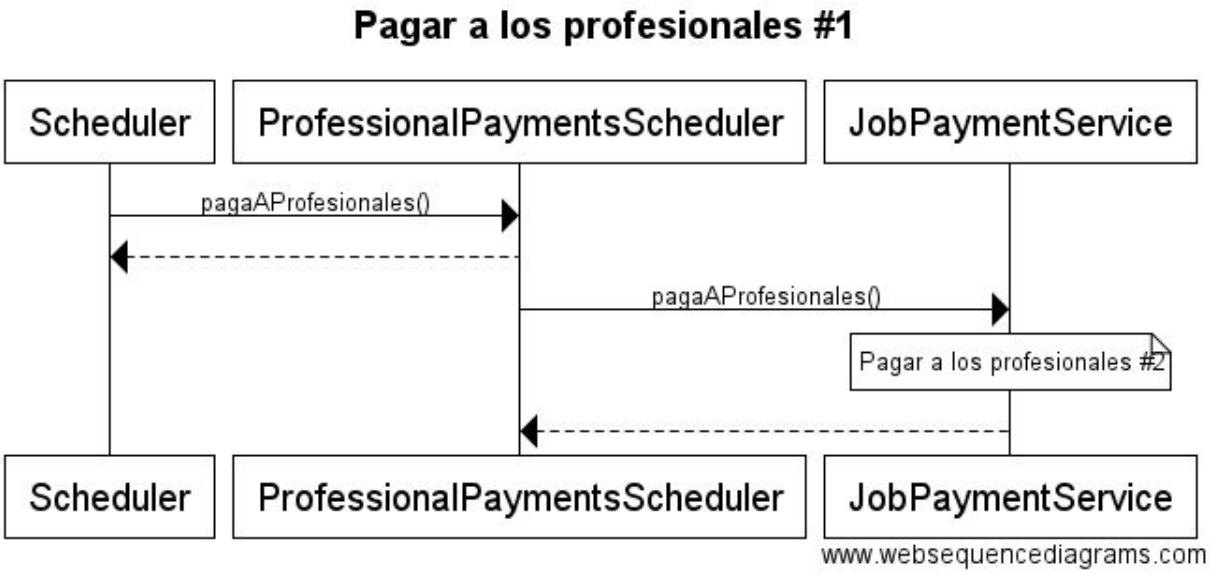
7.5.5.2. Pagar tarea



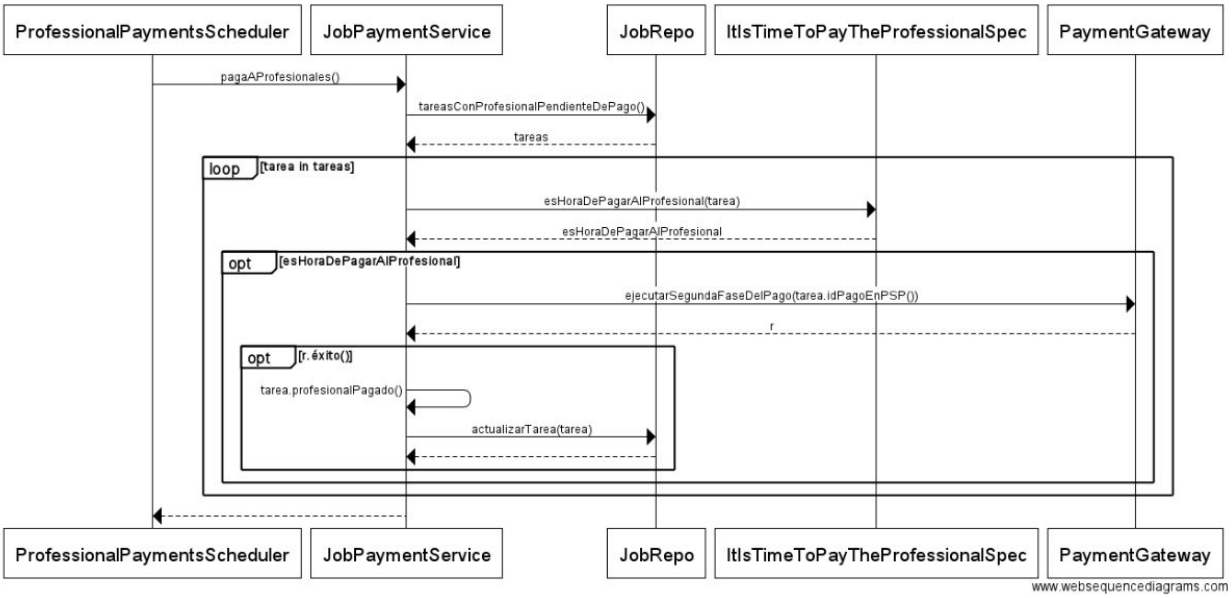


'JP' es la abreviación de 'JobPayment'.

7.5.5.3. Pagar a los profesionales

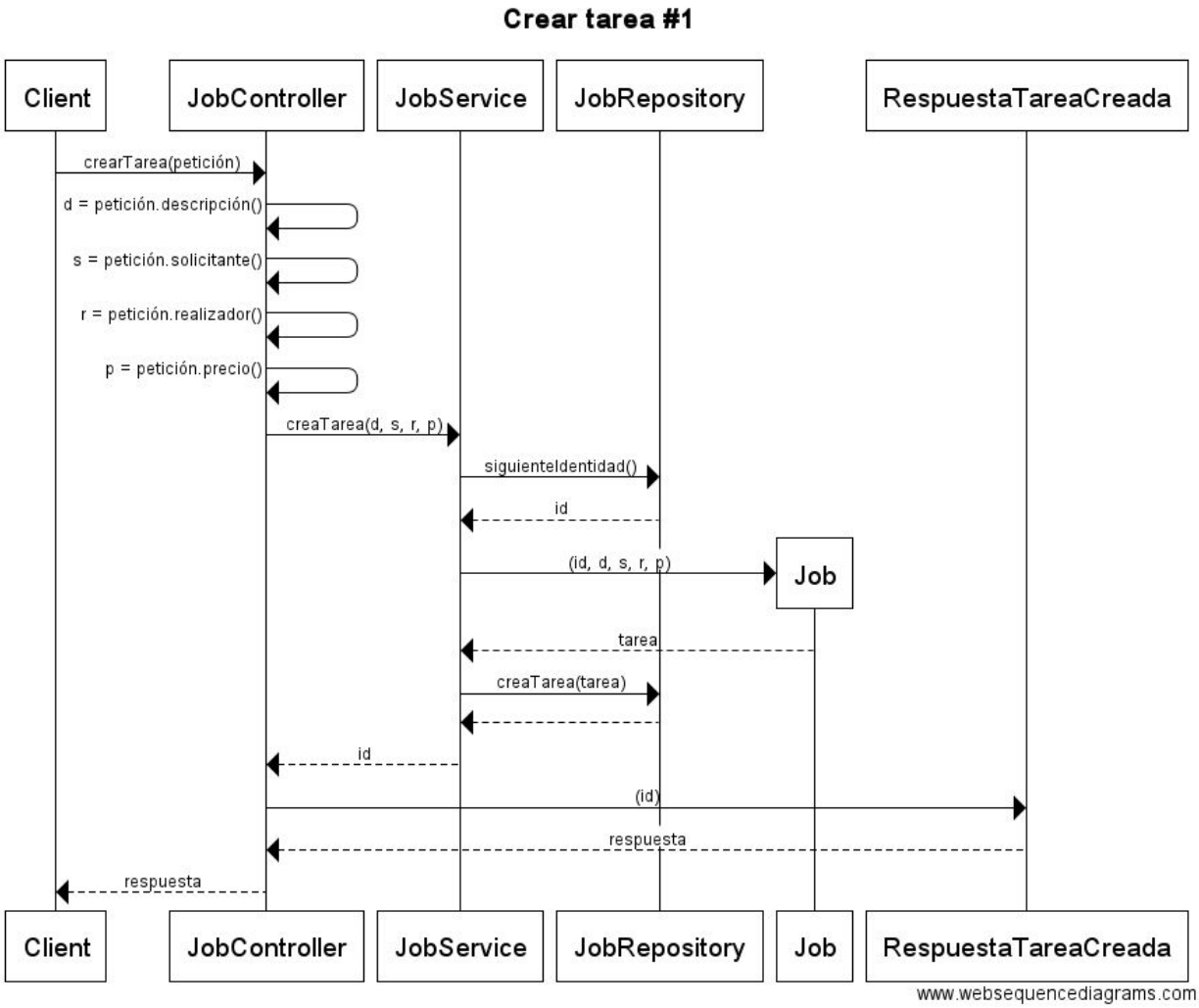


Pagar a los profesionales #2



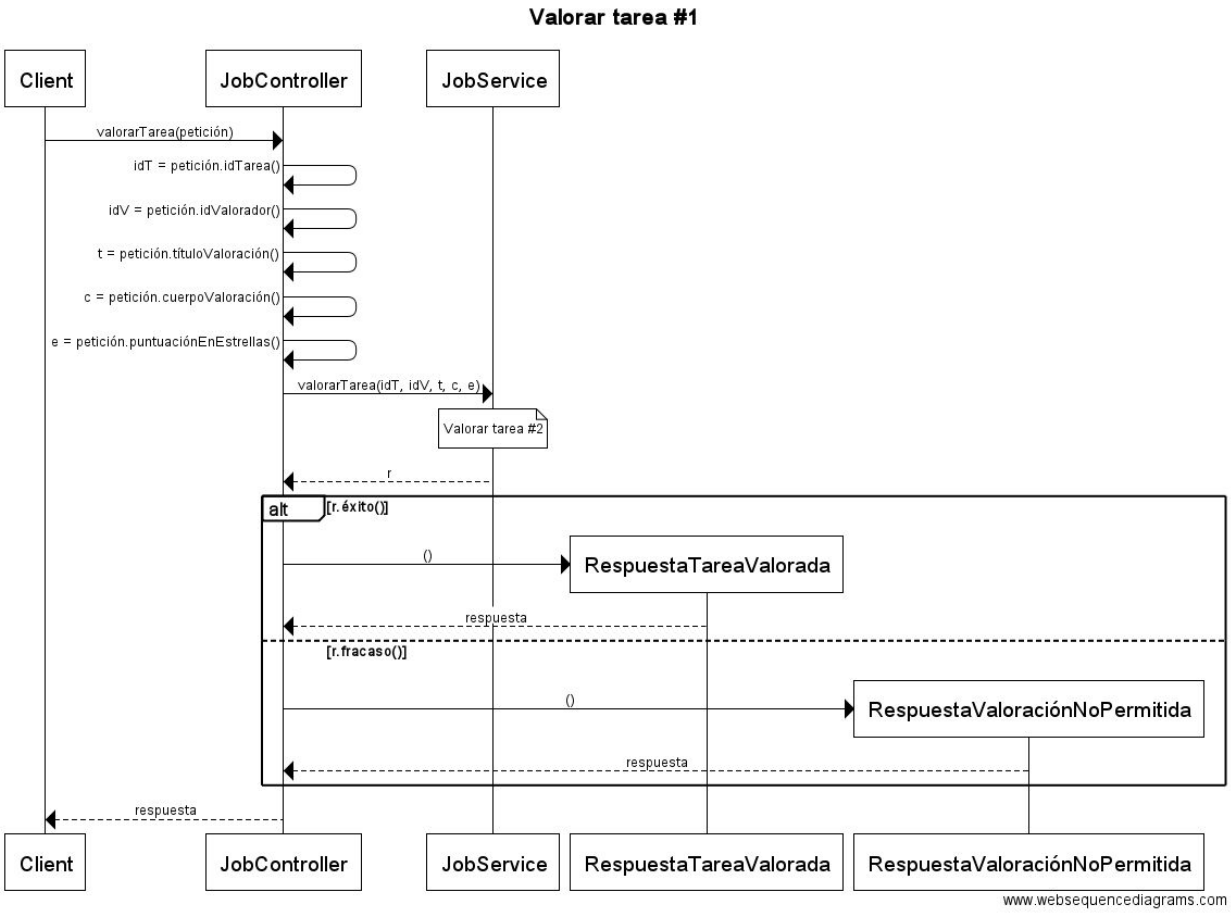
7.5.6. Diagramas de secuencia de tareas

7.5.6.1. Crear tarea

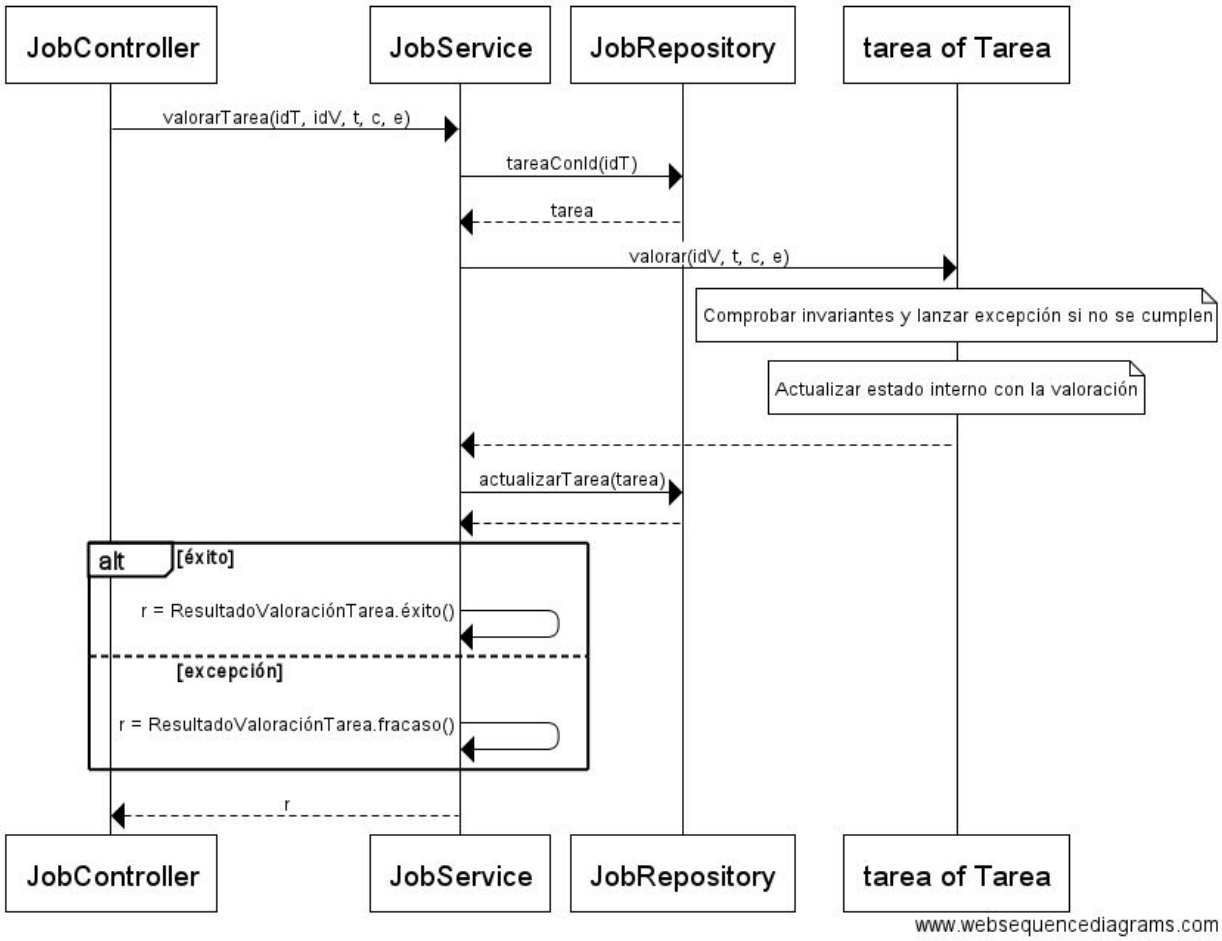


7.5.7. Diagramas de secuencia de valoraciones de tareas

7.5.7.1. Valorar tarea

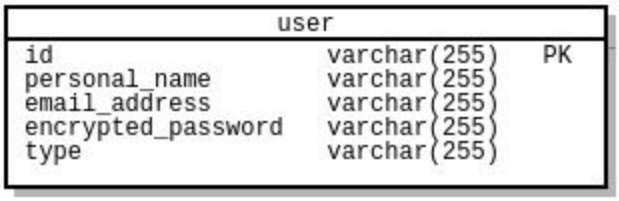


Valorar tarea #2



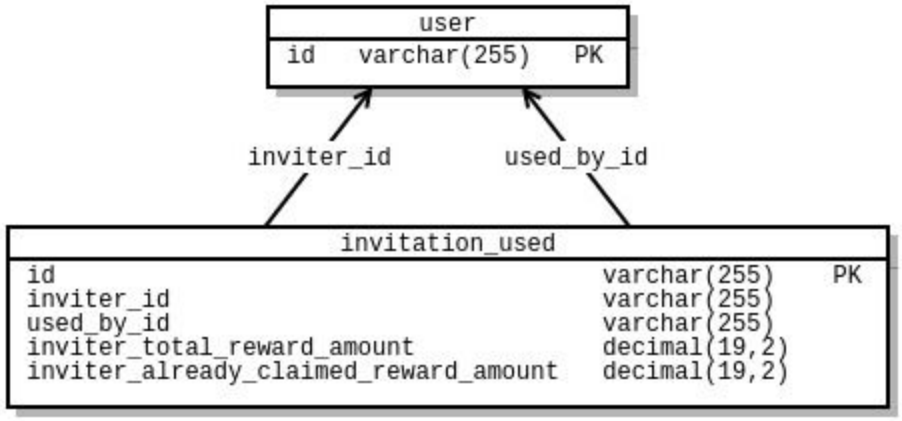
7.6. Diseño de los datos: diagramas de bases de datos

7.6.1. Base de datos de identidad y acceso



La tabla ‘user’ contiene a todos los usuarios del sistema, con su dirección de correo electrónico, su contraseña encriptada, su nombre personal y su tipo (cliente o profesional).

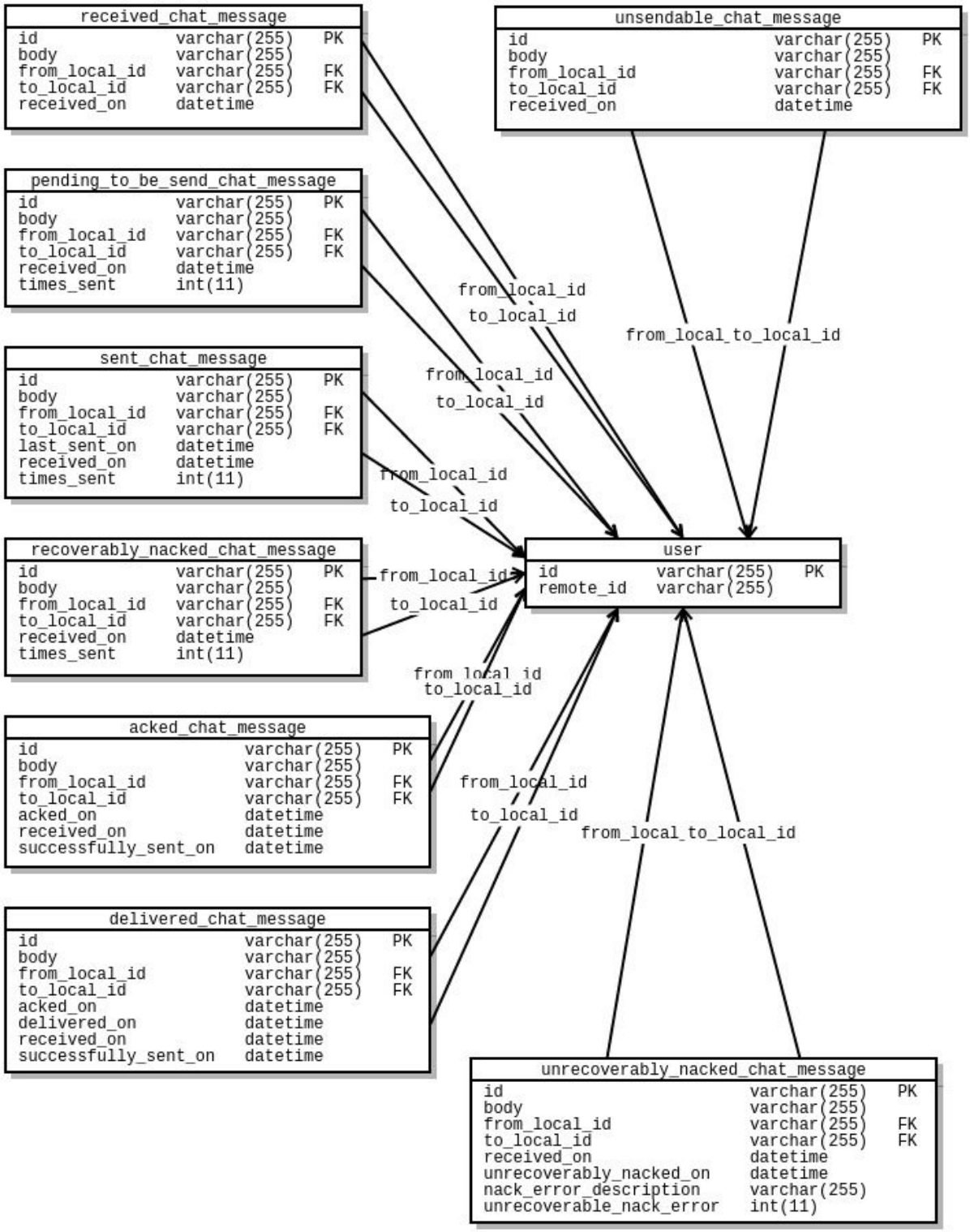
7.6.2. Base de datos de invitaciones



Por un lado, la tabla ‘user’ contiene solamente identificadores de usuarios que, o bien han usado alguna invitación, o bien han mandado alguna invitación que ha sido usada.

Por otro lado, la tabla ‘invitation_used’ contiene información acerca de las invitaciones que han sido usadas: el identificador del invitador, el identificador del invitado, la cantidad con la que se premió al invitador, y la cantidad del total del premio que ya ha sido reclamada por el invitador.

7.6.3. Base de datos de mensajería instantánea


received_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
received_on datetime
unsendable_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
received_on datetime
pending_to_be_send_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
received_on datetime
times_sent int(11)
sent_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
last_sent_on datetime
received_on datetime
times_sent int(11)
recoverably_nacked_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
received_on datetime
times_sent int(11)
user
id varchar(255) PK
remote_id varchar(255)
acked_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
acked_on datetime
received_on datetime
successfully_sent_on datetime
delivered_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
acked_on datetime
delivered_on datetime
received_on datetime
successfully_sent_on datetime
unrecoverably_nacked_chat_message
id varchar(255) PK
body varchar(255)
from_local_id varchar(255) FK
to_local_id varchar(255) FK
received_on datetime
unrecoverably_nacked_on datetime
nack_error_description varchar(255)
unrecoverable_nack_error int(11)
from_local_id
to_local_id
from_local_to_local_id
from_local_id
to_local_id
from_local_id
to_local_id
from_local_id
to_local_id
from_local_id
to_local_id
from_local_id
to_local_id
from_local_to_local_id

Todas las tablas de mensaje de chat contienen información común: el identificador del mensaje de chat, el identificador local del usuario que ha enviado el mensaje, el identificador local del usuario que ha recibido el mensaje, la fecha en que recibimos el mensaje en nuestro servidor y el cuerpo del mensaje. Nótese que diferenciamos entre el identificador local del usuario y el identificador remoto: el identificador remoto es el que usa internamente Firebase Cloud Messaging para identificar a los usuarios, mientras que el identificador local es el que usamos nosotros.

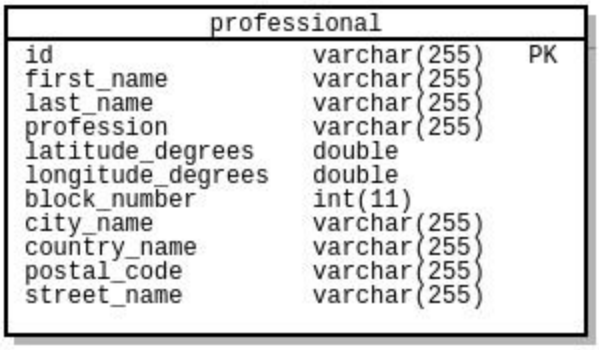
La tabla ‘user’ contiene la tabla de equivalencias entre identificadores locales e identificadores remotos.

Existe una tabla para cada estado en que puede encontrarse un mensaje.

Las columnas ‘sent_on’, ‘acked_on’ y ‘delivered_on’ contienen el momento en que el mensaje de chat fue enviado a uno de los servidores de FCM, el momento en que este servidor de FCM nos confirma que ha recibido el mensaje, y el momento en que este mismo servidor de FCM nos confirma que el mensaje ha sido entregado al dispositivo.

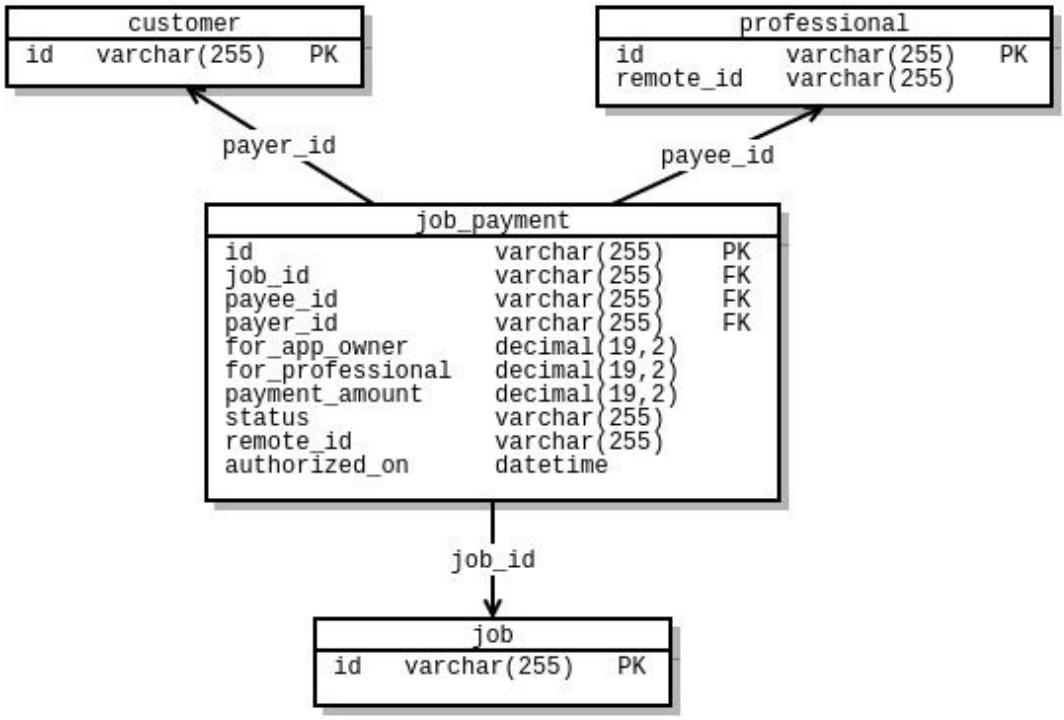
La columna ‘times_sent’ contiene información acerca del número de veces que se ha enviado un mensaje de chat a los servidores de FCM. Téngase en cuenta que puede ser que como respuesta a un envío obtengamos una confirmación negativa de las que permiten volver a intentar el envío.

7.6.4. Base de datos de geodirectorio de profesionales



La tabla ‘professional’ contiene el nombre, el apellido, la profesión y los datos de ubicación de todos los profesionales de la aplicación.

7.6.5. Base de datos de pagos de tareas



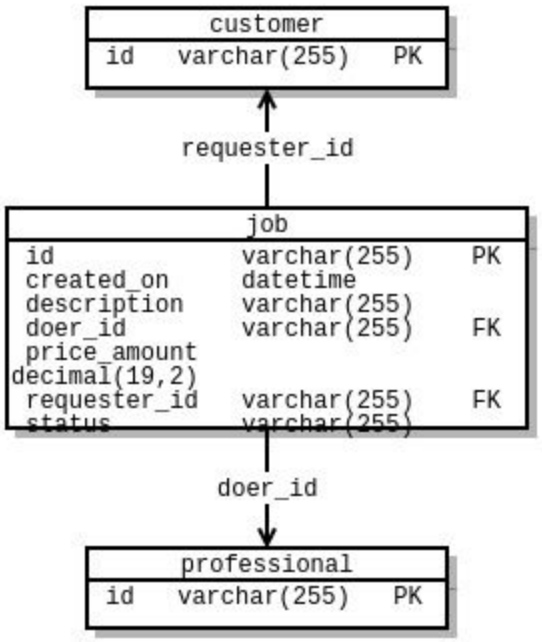
En primer lugar, la tabla ‘job_payment’ contiene la información referente a todos los pagos de tareas que se han intentado en el sistema. Además de contener el identificador del cliente pagador y del profesional pagado, también contiene la distribución del pago, esto es, qué parte se llevará el profesional y qué parte el propietario, e información sobre si el pago ha sido autorizado y, en caso afirmativo, en qué momento se produjo la autorización.

En segundo lugar, la tabla ‘profesional’ contiene la tabla de equivalencia entre los identificadores que usamos localmente para los profesionales, y los identificadores que usa el

proveedor de pagos para esas mismas personas. Este identificador remoto es necesario para indicarle al proveedor de pagos el destinatario del pago.

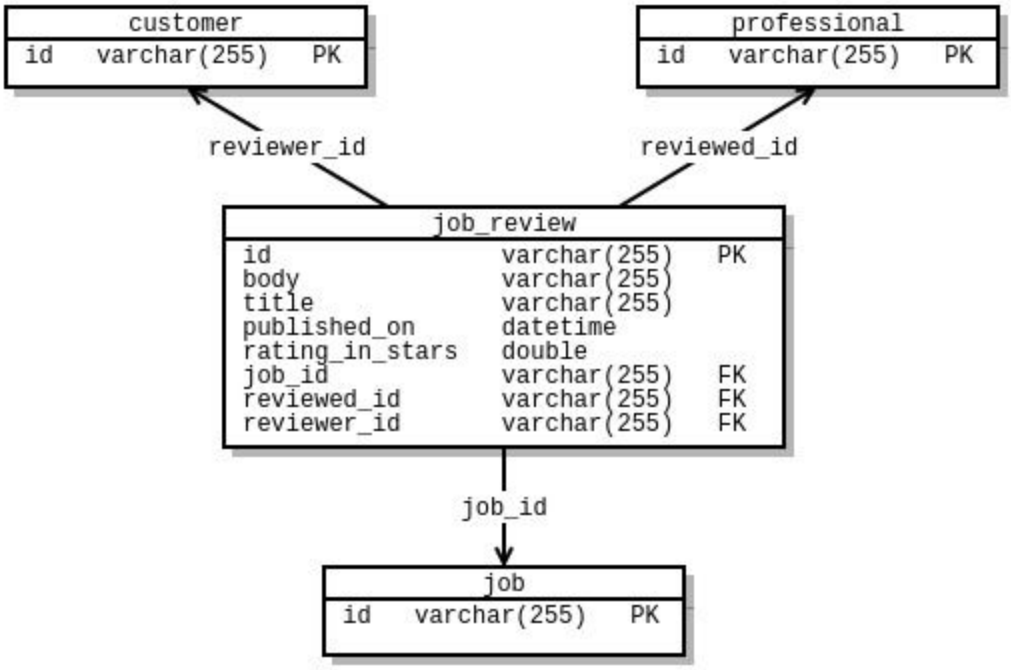
Por último, la tabla ‘job’ solamente contiene identificadores de tareas que han sido pagadas.

7.6.6. Base de datos de tareas



La tabla ‘job’ contiene información sobre todas las tareas que se han creado en el sistema, con información de quién la solicita, de a quién se solicita, del precio y del estado.

7.6.7. Base de datos de valoraciones de tareas



La tabla ‘job_review’ contiene información sobre las valoraciones de tareas en el sistema: título de la valoración, cuerpo de la valoración, fecha de publicación, puntuación en estrellas, tarea a la que se refiere, profesional a quien se valora, y cliente que valora.

7.7. Discusión profunda sobre ciertas decisiones de diseño

En esta sección, lo que haré es profundizar en los aspectos del diseño que he considerado más interesantes. A cada uno de estos aspectos le dedicaré una subsección.

7.7.1. Publicación de y suscripción a eventos dentro de un modelo del dominio

Es importante mantener el dominio libre de elementos de infraestructura, por eso los sistemas de mensajería típicos que utilizamos para comunicar eventos entre contextos delimitados diferentes no nos sirven para publicar o suscribirnos a eventos del dominio.

Aunque existen varias maneras de hacer esto posible, la variante que sugiere implementar el patrón Observador, de modo que los suscriptores sean los observadores y el publicador el notificador, es probablemente la que goza de mayor popularidad actualmente. Una implementación típica es la que nos presenta Vaughn Vernon en la página 299 de su libro *Implementing Domain-Driven Design*, y que reproduzco a continuación:

```
public class DomainEventPublisher {

    @SuppressWarnings("unchecked")
            private static final ThreadLocal<List> subscribers = new
ThreadLocal<List>();

           private static final ThreadLocal<Boolean> publishing = new
```

```
ThreadLocal<Boolean>() {
        protected Boolean initialValue() {
            return Boolean.FALSE;
        }
    };

    public static DomainEventPublisher instance() {
        return new DomainEventPublisher();
    }

    public DomainEventPublisher() {
        super();
    }

    @SuppressWarnings("unchecked")
    public <T extends DomainEvent> void publish(final T domainEvent) {
        if (publishing.get()) {
            return;
        }
        try {
            publishing.set(Boolean.TRUE);
                        List<DomainEventSubscriber<T>> registeredSubscribers =
subscribers.get();
            if (registeredSubscribers != null) {
                Class<?> eventType = domainEvent.getClass();
                                for  (DomainEventSubscriber<T> subscriber  :
registeredSubscribers) {
```

```
                                                    Class<?>  subscribedTo  =
subscriber.getSubscribedToEventType();
                                if (subscribedTo == eventType || subscribedTo ==
DomainEvent.class) {
                            subscriber.handleEvent(domainEvent);
                        }
                    }
                }
            } finally {
                publishing.set(Boolean.FALSE);
            }
        }

        public DomainEventPublisher reset() {
            if (!publishing.get()) {
                subscribers.set(null);
            }
            return this;
        }

        @SuppressWarnings("unchecked")
          public <T extends DomainEvent> void subscribe(DomainEventSubscriber<T>
subscriber) {
            if (publishing.get()) {
                return;
            }
                        List<DomainEventSubscriber<T>>  registeredSubscribers  =
```

```
subscribers.get();
        if (registeredSubscribers == null) {
            registeredSubscribers = new ArrayList<DomainEventSubscriber<T>>();
            subscribers.set(registeredSubscribers);
        }
        registeredSubscribers.add(subscriber);
    }
}
```

Dado que esta implementación pretende ser simple, es fácil proponer mejoras. Con el código actual, el publicador solamente notifica el evento a aquellos suscriptores que estén suscritos a dicho evento o que estén suscritos a todos los eventos. Pero no permite, por ejemplo, que un suscriptor esté suscrito a dos eventos particulares. Tampoco da la opción de suscribirse a todos los eventos que extiendan determinado evento, lo cuál sería útil en dominios con complejas jerarquías de eventos.

El publicador es una instancia única y tiene estado mutable, de modo que hay que tener cuidado a la hora de utilizarlo en entornos concurrentes. Aquí se supone que los hilos de ejecución realizan operaciones completamente independientes, de modo que podemos guardar el estado del publicador en el almacenamiento local del hilo. Este patrón, almacenamiento local al hilo o Thread-Local Storage, se usa mucho en entornos concurrentes, y diría que fue propuesto por primera vez por Douglas Schmidt y Michael Stal en su libro Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects, publicado en septiembre de 2000. En aquel tomo, lo bautizaron con un nombre diferente: almacenamiento

específico del hilo o Thread-Specific Storage. Douglas Schmidt y Michael Stal son colaboradores de Frank Buschmann, una de las personas más prolíficas en el ámbito de los patrones para el diseño software.

La aplicación de dicho patrón funciona bien si los hilos de ejecución se crean para la realización de una tarea, y se destruyen cuando finaliza esta. El problema se presenta cuando los hilos de ejecución no se destruyen sino que se reaprovechan, como ocurre típicamente en un servidor de aplicaciones que hace uso de un pool de threads. Obviamente cuando estamos ejecutando una operación y publicamos un evento, no queremos notificar a suscriptores no registrados por esta operación, sino que han sido registrados por operaciones anteriores.

En estos casos lo que hay que hacer es resetear el publicador. Hay varias opciones, el uso de aspectos, de la programación orientada a aspectos, se presenta bastante polivalente. En el caso de que atendamos peticiones en un servidor, lo que podemos hacer es interceptar todas las peticiones y aprovechar la intercepción para resetear el estado del publicador.

Los suscriptores no tienen mayor misterio, deben implementar la interfaz de suscriptor especificando como tipo genérico el evento al que desean ser suscritos, lo cual se reduce a especificar el comportamiento del suscriptor cuando recibe uno de esos eventos. Esta simplicidad hace que sea habitual la implementación anónima de suscriptores.

Por último, aprovecho para aclarar que, evidentemente, todo este proceso de publicación y trata de eventos es síncrono. Cuando se publica un evento, el publicador recorre su lista de suscriptores y notifica a aquellos suscritos a dicho evento. La notificación por supuesto es síncrona: el suscriptor trata el evento y vuelve. Cuando termina de recorrer su lista de suscriptores, el publicador vuelve a la sentencia que sigue a la de publicación del evento. Todo dentro del mismo hilo.

No es lo mismo que la publicación de eventos a través de un sistema de mensajería. En ellos, el publicador da la orden de publicar un evento síncronamente, pero cuando los suscriptores reciban el evento crearán un hilo de ejecución específicamente para tratarlo.

7.7.2. Comunicación de eventos a otros sistemas por mensajería; suscripción a eventos de otros sistemas por mensajería

Una vez visto cómo se gestiona la publicación y la suscripción de eventos dentro de un mismo sistema, veamos ahora cómo comunicar eventos ocurridos en un sistema a otros sistemas, y cómo suscribir un sistema a eventos ocurridos en otros sistemas.

Hay muchas maneras de lograr esto, probablemente tantas como formas hay de integrar sistemas. Mis opciones preferidas son dos: o bien publicar los eventos como recursos REST, o bien enviar los eventos como mensajes a través de un intercambiador o tema de un middleware orientado a mensajes.

Con cualquiera de estas dos alternativas logramos dos cosas: sistemas desacoplados y sistemas autónomos. Por un lado, logramos sistemas desacoplados puesto que los sistemas publican eventos sin saber nada acerca de los sistemas suscritos a dichos eventos. Por otro lado, logramos sistemas autónomos puesto que los sistemas no necesitan para publicar eventos que los sistemas suscritos a dichos eventos estén funcionando, ni necesitan para suscribirse a eventos que los sistemas publicadores de dichos eventos estén funcionando.

El contenido de los eventos es siempre muy heterogéneo, y cuando un sistema publica dos o más tipos de evento, lo que no hace es publicar cada tipo de evento como un recurso diferente, ni tampoco enviar cada tipo de evento a través de un intercambiador o tema diferente. Hacer eso sería excesivo tanto para el sistema publicador como para los sistemas suscritos. Además, a un sistema suscrito a eventos de varios publicadores distintos, también le facilitará las cosas que todos los publicadores publiquen eventos con el mismo formato. Mi propuesta para la

publicación de eventos siguiendo un formato común es el tipo notificación, cuya implementación en Java detallo a continuación:

```
public class Notification {
      private Long eventId;
      private Date ocurredOn;
      private String eventType;
      private String eventBody;
      ...
}
```

El tipo de evento permite a los suscriptores filtrar los eventos que le llegan de un publicador y quedarse solamente con aquellos que le interesan. El cuerpo del evento contiene la serialización del contenido del evento: el suscriptor interesado debería ser capaz de deserializar y recuperar el contenido del evento.

El contenido de los eventos, aunque debería ser estable, de cuando en cuando cambia. Igual que las APIs. Tanto los eventos que publica un sistema como sus APIs deberían estar documentados en el lenguaje compartido por el sistema con sus clientes. Sin embargo, los clientes no están constantemente pendientes de la documentación para observar si hay cambios, e incluso hay clientes que son sistemas legados y que ya no se actualizan.

Por estos motivos, la actualización del contenido de los eventos podría romper cosas. Igual que la actualización de una API podría romper cosas. Para evitar que un sistema suscrito a un tipo

de evento de otro sistema deje de funcionar cuando ese otro sistema actualice el contenido de ese tipo de evento, lo que se puede hacer es versionado de eventos, igual que los sistemas versionan sus APIs para evitar que muchos clientes dejen de funcionar cada vez que se publica una nueva versión de la API.

El versionado de eventos es un asunto complejo: no basta con incluir la versión del evento en el tipo notificación. De modo que mi propuesta aquí es diferente: hay que mantener a los suscriptores tan desacoplados del contenido del evento como sea posible. Con esto quiero decir que los suscriptores deberían procesar solamente aquellos campos del contenido del mensaje que les son absolutamente imprescindibles. De esta forma lograríamos que en vez de estar acoplados a todo el contenido del mensaje, estuvieran solamente acoplados a una parte de él.

Esto suele funcionar en la mayoría de los casos por la siguiente razón: los publicadores evitan alterar o eliminar los campos que contenía un tipo de evento originalmente, porque saben que es probable que existan suscriptores que hagan uso de esos campos. Lo que sí hacen de tanto en tanto es modificar el contenido del evento para añadir información, para añadir nuevos campos. Esta decisión la toma el equipo de desarrollo del sistema publicador, bien sea por iniciativa propia y sin que nadie se lo pida, bien sea porque se lo ha pedido el equipo de desarrollo de alguno de los sistemas suscriptores.

Así pues, un sistema suscrito a un tipo de evento que solamente está acoplado a las propiedades que usa efectivamente, no tendrá problema cuando se le añadan nuevos campos a ese tipo de evento. En otro caso, los nuevos elementos extraños en el contenido del evento podrían imposibilitar su deserialización.

Y aquí viene la otra recomendación: que el formato del serializado permita leer campos o propiedades sin necesidad de tener que deserializar todo el objeto. Los formatos JSON y XML por ejemplo funcionan bien, y de tener que elegir uno me quedaría con el primero porque es más ligero. Los Protocol Buffers de Google también son una buena opción, ya que incluso permitirían alterar el nombre de los campos de un evento sin causar problemas al suscriptor, puesto que este método de serialización se basa en la posición de los campos o propiedades en vez de en sus nombres.

Una vez tratados estos asuntos de bajo nivel, volvamos a la cuestión del asunto. En la implementación se ha optado por hacer uso de mensajería para comunicar eventos entre sistemas o contextos delimitados y eso es lo que vamos a tratar. Este tipo de integración por mensajería se puede implementar de múltiples maneras, de modo que me centraré en mi propuesta.

Comenzaremos por un análisis de alto nivel para luego pasar a los detalles. Cada contexto delimitado o sistema hace uso de su propio intercambiador o tema dentro del middleware orientado a mensajes para enviar mensajes. Estos mensajes serán notificaciones de eventos. Los contextos delimitados o sistemas suscriptores tendrán su propio cola, la cual vincularán a todos los intercambiadores de aquellos sistemas de los cuales deseen recibir notificaciones de eventos. Además, deberán conocer el tipo de aquellos eventos en los que estén interesados, para poder filtrar y procesar.

En la práctica, para evitar que la clase que gestiona la llegada de eventos se convierta en un objeto todopoderoso o God object, lo que se hace es dividir esa lógica en múltiples clases, cada una de las cuales se encargará de gestionar la llegada de un tipo evento. Cada una de estas clases suscriptor también hará uso de su propia cola, que por conveniencia puede ser el

propio nombre de la clase. Esta es la forma de garantizar que cada uno de estos pequeños suscriptores recibe todos los eventos del tipo que le interesa. Los intercambiadores, por supuesto, publicarán las notificaciones de eventos a todas las colas de todos los pequeños suscriptores de todos los sistemas suscritos.

Una vez tomado este camino, es sencillo ver que todos esos pequeños suscriptores tienen mucho en común: todos configurarán su cola con un consumidor de mensajes que procesará solamente aquellos mensajes que sean notificaciones de eventos del tipo que les interesa, y todos vincularán su cola al intercambiador del sistema que publica dichos eventos.

En otras palabras, lo único que tiene que hacer un suscriptor es, por un lado, especificar el intercambio al que quiere vincular su cola, y, por otro lado, implementar la lógica de reacción al evento. Toda la configuración la puede realizar una clase padre que consulte estos detalles a la clase hija, cualquiera de los pequeños suscriptores.

Así lo entiende, por ejemplo, Vaughn Vernon. En su libro Implementing Domain-Driven Design, incluye una posible implementación de esta clase padre en las páginas 471 y 472:

```
public abstract class ExchangeListener {


      private MessageConsumer messageConsumer;

      private Queue queue;
```

```
	public ExchangeListener() {
	super();

	this.attachToQueue();

	this.registerConsumer();
	}

	protected abstract String exchangeName();

	protected abstract void filteredDispatch(String aType, String
aTextMessage);

	protected abstract String[] listensToEvents();

	protected String queueName() {
	return this.getClass().getSimpleName();
	}
```

```
private void attachToQueue() {

Exchange exchange =

      Exchange.fanOutInstance(

                  ConnectionSettings.instance(),

                  this.exchangeName(),

                  true);



this.queue =

      Queue.individualExchangeSubscriberInstance(

                  exchange,

                  this.exchangeName() + "." + this.queueName());

}



private Queue queue() {

return this.queue;

}



private void registerConsumer() {

this.messageConsumer = MessageConsumer.instance(this.queue(), false);
```

```
        this.messageConsumer.receiveOnly(
                this.listensToEvents(),
                new MessageListener(MessageListener.Type.TEXT) {


                @Override
                public void handleMessage(
                        String aType,
                        String aMessageId,
                        Date aTimestamp,
                        String aTextMessage,
                        long aDeliveryTag,
                        boolean isRedelivery)
                throws Exception {
                filteredDispatch(aType, aTextMessage);
                }
        });
        }
}
```

Y aquí uno de mis pequeños suscriptores que hace uso de esta clase padre:

```
public class RabbitMQJobPaidListener extends ExchangeListener {

    private JobApplicationService jobApplicationService;


                    protected     RabbitMQJobPaidListener(JobApplicationService
jobApplicationService) {

       this.jobApplicationService = jobApplicationService;

    }


    @Override

    protected String exchangeName() {

       return Exchanges.JOB_PAYMENTS_EXCHANGE_NAME;

    }


    @Override

    protected void filteredDispatch(String aType, String aTextMessage) {

       NotificationReader reader = new NotificationReader(aTextMessage);
```

```
        String jobId = reader.eventStringValue("jobId.id");


        this.jobApplicationService.jobPaid(jobId);
    }


    @Override
    protected String[] listensTo() {
        return new String[] {
              "com.marketplace.jobpayments.domain.jobpayment.jobPaid"
              };
    }
}
```

Cabe aclarar que el lector de notificaciones o NotificationReader es el objeto que me permite leer las propiedades o campos de un objeto serializado sin que sea necesario deserializar el objeto por completo. También cabe aclarar que el Sr. Vernon permite que los pequeños suscriptores estén suscritos a más de un evento: en esto difiere de mi visión de un evento por suscriptor.

Cuando utilizamos mensajería para notificar los eventos a sistemas remotos, queremos asegurarnos de que los eventos sean recibidos al menos una vez, en contraposición a que sean recibidos como mucho una vez. En este sentido ayuda mucho que la lógica de tratamiento del evento sea idempotente, esto es, que el efecto sobre el sistema sea el mismo se trate el evento una o varias veces.

La parte de publicar notificaciones de eventos es más sencilla. De lo primero que tiene que encargarse un sistema que notifica sus eventos es de no perderse ningún evento. Para ello lo que hará es guardar todos los eventos que se produzcan durante la ejecución de los diferentes casos de uso.

Ahora bien, ya que tenemos que hacerlo, optemos por la vía más fácil: usemos una misma base de datos para guardar el estado del dominio y los eventos. Así podemos realizar el guardado de eventos de un caso de uso dentro de la misma transacción que el caso de uso, y esto es una forma muy sencilla de garantizar que no se pierde ningún evento. Efectivamente, perder un solo evento podría dar lugar a inconsistencias importantes entre diferentes sistemas, por eso es tan importante incidir en este tema.

Los eventos se almacenan en lo que se conoce como almacén de eventos o Event Store. Ahí los eventos se guardan como notificaciones, con el contenido serializado, de modo que dentro de una misma estructura de almacenamiento se pueden guardar todo tipo de eventos, pues una vez serializados todos tendrán un formato común.

Vamos ahora con el servicio que publica las notificaciones de eventos. Este servicio implementará el caso de uso de publicación de notificaciones. Lo que hará es recuperar del almacén de eventos todos aquellos eventos que todavía no han sido notificados, y los publicará

como notificaciones de evento. Es decir, los enviará como mensajes a través de su intercambiador o tema.

Para saber qué eventos han sido publicados y cuáles no, hay dos opciones. La primera, la menos buena, consiste en añadir a los eventos almacenados en el almacén de eventos una bandera o flag que indique si el mensaje ha sido publicado o no. En este caso el servicio seleccionaría todos los eventos del almacén de eventos con esta bandera a falso, los publicaría, y los guardaría con la bandera a cierto. Todo dentro de la misma transacción.

La segunda opción se beneficia de la existencia de esta transacción. Asumamos dos cosas: que el almacén de eventos asigna un número que se va incrementando como identificador a los eventos que van llegando, y que tenemos almacenado en la base de datos un objeto que contiene el identificador del último evento publicado. La transacción debería incluir también, por supuesto, el guardado en base de datos del objeto con el identificador del último evento publicado.

Siendo así, podemos asegurar dos cosas: que todos los eventos con un identificador inferior o igual al identificador del último evento publicado han sido publicados, y que todos los eventos con un identificador superior al identificador del último evento publicado no han sido publicados.

En efecto, es fácil ver que el identificador del último evento publicado no puede ser superior al identificador de un evento no publicado. Dicho evento no publicado, en algún momento debió llegar al almacén de eventos, donde se le asignó un identificador que era en aquel momento superior al del último evento publicado. Desde entonces, o bien todos los subsiguientes intentos de publicación de eventos fallan al publicar este evento, con lo cual no se actualiza el identificador del último evento publicado, o bien alguno de ellos tiene éxito al conseguir publicar este evento y todos los pendientes, con lo cual se actualiza el identificador del último evento

publicado. Pero lo que no es posible es que se falle al publicar un evento, y sin embargo se actualice el identificador del último evento publicado.

De modo que la segunda opción consiste precisamente en esto: en un almacén de eventos que asigne como identificador un número que se va incrementando a los eventos que van llegando, y en un objeto guardado en la base de datos con el identificador del último evento publicado. De esta forma el publicador de notificaciones de eventos puede obtener aquellos eventos que no han sido publicados y publicarlos.

Efectivamente, lo que tendría que hacer es obtener el objeto con el identificador del último evento publicado de la base de datos, obtener todos los eventos del almacén de eventos con un identificador superior al identificador del último evento publicado, publicar dichos eventos, actualizar el identificador del último evento publicado, y guardar el objeto que lo contiene en base de datos. Todo dentro de la misma transacción.

En caso de fallar la transacción, los eventos publicados con éxito hasta el momento del fallo siguen su camino hasta sus respectivos suscriptores. La forma de evitarlo sería realizar una transacción distribuida, que incluyera los cambios en la base de datos local así como los cambios en la base de datos del bróker de mensajería, y que ejecutara el commit en dos fases. Debido a la complejidad intrínseca de las transacciones distribuidas, y a que no son demasiado eficientes, este es un recurso que prefiero no utilizar.

Así pues, los eventos publicados con éxito durante una transacción fallida serán publicados, como mínimo, una vez más: cuando la transacción sea un éxito. De ahí la importancia de que el tratamiento de los eventos recibidos por parte de los suscriptores sea idempotente, en la medida de lo posible, tal y como he comentado más arriba en esta misma sección.

Lo único que falta es que alguien se encargue de publicar periódicamente los eventos pendientes de ser publicados. Esto puede hacerlo perfectamente un planificador. Es cuestión de configurar el intervalo entre publicaciones e iniciarlo. Sería interesante tener monitorizado este planificador para poder comprobar que sigue en funcionamiento. Si dejara de funcionar, en poco tiempo la inconsistencia temporal entre el publicador y sus suscriptores se dispararía, y esto sin duda es una situación a evitar.

7.7.3. Cómo modelar los diferentes estados de una entidad

Una opción es utilizar el patrón estado o State Pattern, o incluso el patrón estrategia o Strategy Pattern en algunos casos, ambos muy conocidos desde que se popularizó el libro que los presentó al mundo: *Design Patterns: Elements of Reusable Object-Oriented Software*, Erich Gamma et al., 1994.

El principal inconveniente de este patrón es que su implementación casi siempre resulta en una gran cantidad de código redundante. Por ejemplo, si tenemos definidas ocho transiciones de estado, todos los estados tendrán que implementar las ocho transiciones, incluso aquellos desde los cuales no se puede transicionar a ningún otro: en ese caso la implementación de las ocho transiciones consistirá en lanzar una excepción de estado inválido, o Invalid State Exception.

Otra opción es modelar los estados como un tipo enumerado, y que el tipo entidad contenga todas las propiedades necesarias para ser capaz de representar la entidad en cualquiera de sus posibles estados. Obviamente, cuanto más heterogéneas sean las propiedades (el estado) de los diferentes estados, menos atractiva resulta esta opción. Sin embargo, lo habitual es que los estados tengan muchas propiedades en común. Esto, junto con la versatilidad que otorgan lenguajes de programación como Java a sus tipos enumerados, constituye un gran argumento a favor de esta alternativa.

Existe al menos una tercera opción que vale la pena tratar. Cuando leí a Scott Millett diciendo que había que evitar el patrón estado, y que en vez de eso modeláramos explícitamente, reconozco que me llamó poderosamente la atención. En su momento, todo me parecieron ventajas: los diferentes estados quedan modelados explícitamente, los tipos de los diferentes

estados solamente incluyen las propiedades que se usan efectivamente, es decir, evitamos los nulos, y los tipos de los diferentes estados proporcionan una interfaz limpia, que solamente incluye los métodos de transición que son aplicables.

Decidí aplicarlo en el sistema de mensajería instantánea, con los mensajes de chat. Cuando implementé un primer dominio no tuve problemas, ni siquiera cuando implementé un segundo dominio para adecuarme al modelo de mensajería instantánea de Firebase Cloud Messaging. Sin embargo, cuando me puse con los casos de uso comenzaron los problemas.

Conocer cómo se producen las transiciones de un estado a otro en el dominio ayuda a comprender el origen de estos problemas. El siguiente ejemplo lo ilustra muy bien:

```
public class ReceivedChatMessage extends ChatMessage {
	private Date receivedOn;
	...

	public SentChatMessage sent(Date sentOn) {
		return new SentChatMessage(
				super.id(),
				super.from(),
				super.to(),
				super.body(),
				this.receivedOn(),
				sentOn);
```

```
        }
}
```

En efecto, el mensaje de chat cambia de clase pero preserva su identidad. Cambia de tipo pero preserva su identidad. Este uso del paradigma de la programación orientada a objetos seguramente no estaría bien visto por sus usuarios más puristas.

Cuando hay que persistir este cambio de tipo preservando la identidad en una base de datos relacional, las cosas se complican. Las complicaciones surgen tanto si en el modelo relacional ambos tipos, el origen y el destino, están modelados como tipos diferentes, como si en el modelo relacional ambos tipos están modelados como el mismo tipo.

Por un lado, cuando en el modelo relacional ambos tipos están modelados como tipos diferentes, toda transición de estado conlleva la eliminación de un registro del tipo origen, y la creación de un registro del tipo destino. No resulta intuitivo ver cómo un servicio de aplicación obtiene un objeto del repositorio, lo utiliza para crear otro objeto, elimina el objeto original y guarda el objeto creado.

Por otro lado, cuando en el modelo relacional ambos tipos están modelados como el mismo tipo, las complicaciones son mayores. En este caso, no es posible utilizar un ORM que implemente el patrón unidad de trabajo o Unit of Work. Como se sabe, estos ORMs lo que hacen es acumular los cambios a realizar en la base de datos, y no los realizan efectivamente hasta que no es el momento de hacer commit.

Durante la transacción, lo que indicamos es que se elimine un objeto de un tipo con determinado identificador, y que se cree un objeto de otro tipo con ese mismo identificador. Sin

embargo, como ambos tipos están mapeados al mismo tipo del modelo relacional, lo que realmente estamos indicando es que se cree un objeto de un tipo con cierto identificador, y que se elimine un objeto del mismo tipo con el mismo identificador.

Cuando llega el momento de realizar los cambios en la base de datos, el ORM no tiene cómo garantizar la existencia de un objeto, y la no existencia de ese mismo objeto; la existencia de un registro, y la no existencia de ese mismo registro.

Si los cambios no se acumulan, sino que se ejecutan en la base de datos tan pronto como se realizan, este problema no se produce. En efecto, al hacer commit de la transacción la base de datos aplicará los cambios por orden, y no tendrá problema en eliminar un cierto registro para después recrearlo. Nótese que en este contexto considero iguales dos registros si contienen el mismo valor en su columna identificador o clave primaria, independientemente de los valores que contengan en el resto de columnas.

Eso sí, lo que nunca se podrá hacer si ambos tipos están mapeados al mismo tipo relacional es crear el objeto del tipo destino antes deshacernos del objeto del tipo origen, porque ahí sí que es la propia base de datos la que lo impide, aduciendo que ese registro ya existe (en general, violación de clave primaria).

De modo que el proceder razonable es optar por mapear cada objeto-estado del dominio a una tabla relacional diferente: esto no nos dará problemas ni con el ORM ni con la base de datos. Con dicha opción, decíamos, una transición no se reducía a la actualización de un objeto, sino que se convertía en la eliminación de un objeto y la creación de otro. Además de este inconveniente, existe un problema que se manifiesta cuando tenemos que buscar un objeto sin

conocer su estado. En ese caso habrá que buscar el objeto en el resultado de la unión de tantas tablas como estados pueda tener un objeto. Sin duda, una búsqueda muy costosa.

Ya hemos visto que la tercera opción tampoco está exenta de problemas. Dado que esta es la opción por la que he optado para modelar el dominio del sistema de mensajería instantánea, presentaré ahora un modelo alternativo del mismo dominio a modo de cierre de esta sección, y con ella del apartado.

Este modelo alternativo está basado en la segunda opción presentada en esta sección: modelar los posibles estados que puede adoptar un mensaje de chat como un tipo enumerado, y utilizar un solo tipo para representar cualquier mensaje de chat, en vez de utilizar tipos diferentes para representar mensajes de chat en diferentes estados. Además, representaré el estado interno del tipo mensaje de chat como una secuencia de eventos, utilizando el patrón conocido como Event Sourcing (que viene a significar algo así como nutrirse de los eventos).

Lo cierto es que las dos únicas implementaciones que he visto del patrón Event Sourcing estaban escritas en C#. Esto no es casualidad: C#, a diferencia de Java, permite el uso de un tipo dinámico dentro de un sistema de tipado estático, e incluso permite convertir cualquier objeto al tipo dinámico.

Estas facilidades que nos brinda C# resultan muy útiles cuando hay que reconstruir el estado de un agregado a partir de una secuencia de eventos. El agregado conoce qué tipos de evento le son aplicables, y cómo mutar su estado cuando se le aplica un evento de un tipo que le es aplicable. A efectos de implementación, consideraremos que la lógica de mutación del propio estado que corresponde a la aplicación de un evento de determinado tipo está encapsulada en un método cuyo único parámetro es el evento que se aplica.

Ilustremos qué quiere decir esto volviendo al caso que nos ocupa: el sistema de mensajería instantánea. El agregado en este caso sería el mensaje de chat, cuyo estado debe ser reconstruido a partir de una secuencia de eventos. El estado del mensaje de chat lo conformarían el identificador, el remitente, el destinatario, el contenido del mensaje, y las fechas de recepción, envío, confirmación, entrega y lectura. Los eventos aplicables al mensaje de chat serían mensaje de chat creado, mensaje de chat recibido, mensaje de chat enviado, mensaje de chat confirmado, mensaje de chat entregado y mensaje de chat leído.

La aplicación de cada uno de estos eventos mutaría el estado interno del agregado mensaje de chat de diferente manera. Por ejemplo, el evento mensaje de chat creado contiene información acerca del identificador, del remitente, del destinatario y del contenido del mensaje de chat. Al aplicar este evento, el mensaje de chat cambiará su actual identificador (previsiblemente nulo), por el identificador que contiene el evento; cambiará su actual remitente (previsiblemente nulo), por el remitente que contiene el evento; cambiará su actual destinatario (previsiblemente nulo), por el destinatario que contiene el evento; cambiará su actual contenido (previsiblemente nulo), por el contenido que contiene el evento; y cambiará su actual estado (previsiblemente nulo), por el estado creado. Otro ejemplo: el evento mensaje de chat recibido contiene información acerca de la fecha de recepción. Al aplicar este evento, el mensaje de chat cambiará su actual fecha de recepción (previsiblemente nula), por la fecha de recepción que contiene el evento; y cambiará su actual estado (previsiblemente creado), por el estado recibido.

De modo que el agregado mensaje de chat tendrá un método para aplicar cada uno de los seis eventos que le son aplicables. Estos métodos tendrán un único parámetro: el evento a aplicar.

Por conveniencia, todos los métodos que apliquen eventos se llamarán de la misma forma, y solamente diferirán en el tipo de su único parámetro, que será el tipo del evento a aplicar. Esto nos permitirá hacer uso de la sobrecarga de métodos o method overloading.

Llegados a este punto, el agregado contiene métodos para aplicar todos los tipos de evento que le son aplicables. Sin embargo, lo que le llega por el constructor es una secuencia de eventos de tipo evento, tipo antecesor de todos los tipos de evento del dominio.

¿Cómo obtener un subtipo a partir de un supertipo en una sistema de tipado estático? Probando si el supertipo es de determinado subtipo, y si es así convertir el objeto al subtipo. Necesitamos obtener el subtipo para que se llame el método sobrecargado correcto.

En cambio, C# nos brinda la posibilidad de hacer uso de su tipo dinámico, de modo que bastaría con convertir el supertipo en dinámico al llamar el método sobrecargado, porque entonces el subtipo se determinaría en tiempo de ejecución y con ello se llamaría a la versión correcta del método sobrecargado.

En nuestro caso el supertipo sería evento, y los subtipos serían mensaje de chat creado, mensaje de chat recibido, mensaje de chat enviado, mensaje de chat confirmado, mensaje de chat entregado y mensaje de chat leído.

Y esto es lo que nos permite obtener una implementación tan limpia en C#:

```
public class ChatMessage
{
    private ChatMessageId _chatMessageId;
    private UserId _from;
```

```
private UserId _to;
private ChatMessageBody _body;
private ChatMessageStatus _status;
private DateTime _receivedOn;
private DateTime _sentOn;
private DateTime _confirmedOn;
private DateTime _deliveredOn;
private DateTime _readOn;
private DateTime _sentOn;

public ChatMessage(IEnumerable<DomainEvent> events)
{
    foreach (var @event in events)
    {
        Apply(@event);
    }
}

public void Apply(DomainEvent e)
{
    When((dynamic)e);
}

public void When(ChatMessageCreated e)
{
    _chatMessageId = e.chatMessageId();
    _from = e.from();
```

```
    _to = e.to();
    _body = e.body();
    _status = ChatMessageStatus.Created;
}

public void When(ChatMessageReceived e)
{
    ....
}

public void When(ChatMessageSent e)
{
    ....
}

public void When(ChatMessageConfirmed e)
{
    ....
}

public void When(ChatMessageDelivered e)
{
    ....
}

public void When(ChatMessageRead e)
{
```

```
        ....
    }
}
```

En Java, al carecer de este tipo dinámico, no es viable esta implementación, de modo que tendremos que buscar una implementación alternativa.

Una forma sencilla de hacerlo es aplicar el patrón visitador o Visitor. Los agregados que se nutran de eventos serían los visitadores, mientras que los eventos serían los visitados. La clase evento del dominio, clase antecesora de todos los eventos del dominio, debería aceptar como visitadores a todos los agregados nutridos por eventos del dominio. Los eventos concretos llamarían a un método del agregado, el visitador, y se pasarían ellos mismos como parámetro para ser visitados.

Por tanto, la reconstrucción de uno de estos agregados funcionaría de la siguiente manera. Se iteraría sobre la secuencia de eventos. Para cada evento, se llamaría al método que acepta a ese agregado como visitador. El evento, si es aplicable a dicho agregado, llamará al método del agregado correspondiente para aplicarse como evento, incluyéndose él mismo como parámetro.

Así es como logramos pasar del supertipo al subtipo. La clase evento del dominio quedaría así:

```
public interface DomainEvent {

      public Date ocurredOn();
```

```
	public void applyTo(ChatMessage chatMessage);
}
```

El constructor que reconstruye el estado de una mensaje de chat a partir de una secuencia de eventos quedaría de la siguiente manera:

```
ChatMessage {
	private ChatMessageId id;
	private UserId from;
	private UserId to;
	private ChatMessageBody body;
	private ChatMessageStatus status;
	private Date receivedOn;
	private Date sentOn;
	private Date confirmedOn;
	private Date deliveredOn;
	private Date readOn;

	public ChatMessage(Iterable<DomainEvent> domainEvents) {
		for (DomainEvent domainEvent : domainEvents) {
			domainEvent.applyTo(this);
		}
	}
```

Todos los eventos aplicables a un mensaje de chat implementarían el método para aceptar un mensaje de chat como visitador de la misma manera: llamando al método de mensaje de chat que los aplica, pasándose ellos mismos como parámetro. Incluyo aquí la implementación completa del evento mensaje de chat creado:

```
public class ChatMessageCreated implements DomainEvent {

    private Date ocurredOn;

    private ChatMessageId chatMessageId;

    private UserId from;

    private UserId to;

    private ChatMessageBody body;



    public ChatMessageCreated(

              Date ocurredOn,

              ChatMessageId chatMessageId,

              UserId from,

              UserId to,

              ChatMessageBody body) {
```

```
        this.setOcurredOn(ocurredOn);
        this.setChatMessageId(chatMessageId);
        this.setFrom(from);
        this.setTo(to);
        this.setBody(body);
    }

    @Override
    public Date ocurredOn() {
        return this.ocurredOn;
    }

    @Override
    public void applyTo(ChatMessage chatMessage) {
        chatMessage.when(this);
    }
    ….
}
```

Por último incluyo los métodos que mutan el estado de un mensaje de chat como consecuencia de la aplicación de alguno de los eventos que le son aplicables:

```
public class ChatMessage {
	private ChatMessageId id;
	private UserId from;
	private UserId to;
	private ChatMessageBody body;
	private ChatMessageStatus status;
	private Date receivedOn;
	private Date sentOn;
	private Date confirmedOn;
	private Date deliveredOn;
	private Date readOn;

	...

	public void when(ChatMessageCreated chatMessageCreated) {
		this.setId(chatMessageCreated.chatMessageId());
		this.setFrom(chatMessageCreated.from());
		this.setTo(chatMessageCreated.to());
		this.setBody(chatMessageCreated.body());
		this.setStatus(ChatMessageStatus.CREATED);
	}

	public void when(ChatMessageReceived chatMessageReceived) {
```

```
            ...
        }

        public void when(ChatMessageSent chatMessageSent) {
            ...
        }

        public void when(ChatMessageConfirmed chatMessageConfirmed) {
            ...
        }

        public void when(ChatMessageDelivered chatMessageDelivered) {
            ...
        }

        public void when(ChatMessageRead chatMessageRead) {
            ...
        }
        ...
}
```

Esta sería la solución final.

7.8. Diseño de la interfaz de usuario

Incluyo aquí la interfaz de usuario que correspondería a la aplicación varias entregas después de la actual, también con un motivo de ludificación o gamification.

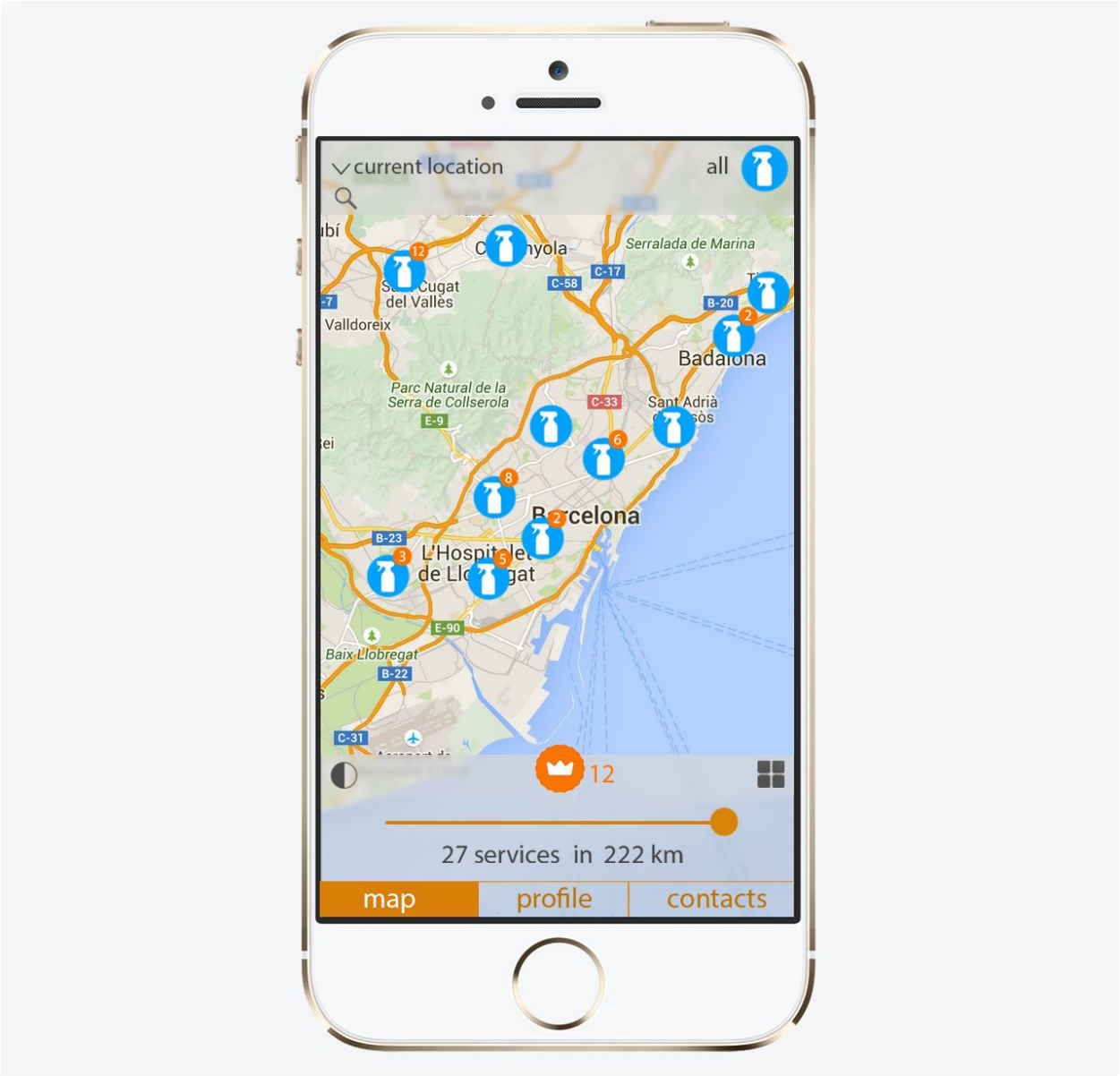
Sobre la ludificación o gamification

La ludificación o gamification consiste en introducir elementos propios de los juegos en contextos diferentes a los juegos: en este caso, en introducir elementos propios de los juegos en nuestra aplicación, que no es un juego. En el caso de este diseño, el elemento propio de los juegos introducido es el sistema de niveles y experiencia.

-

Los diseños se muestran a lo largo de las siguientes páginas.

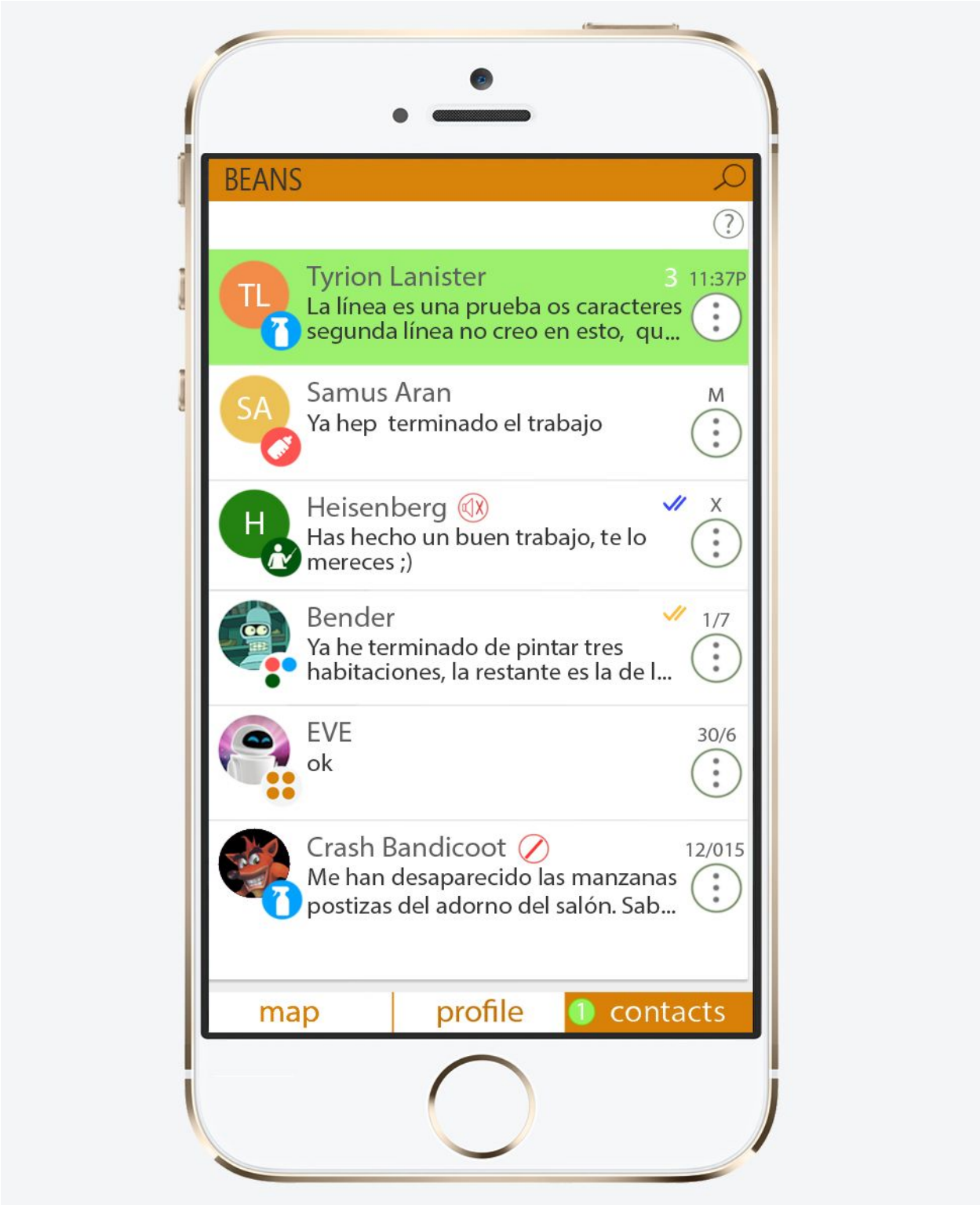
Mapa con los profesionales cercanos de limpieza:



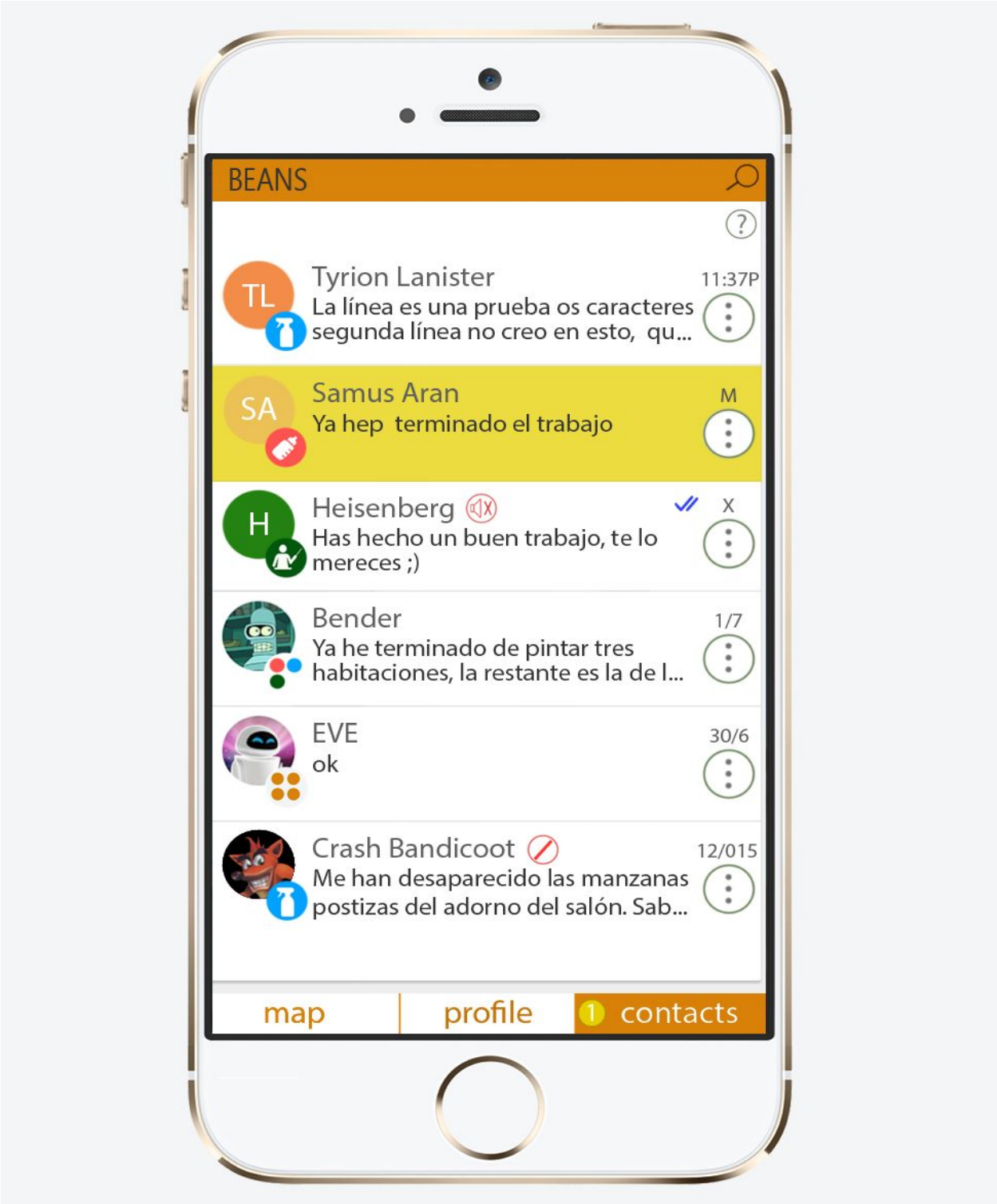
Mapa con todos los profesionales cercanos:

Lo que ocurre cuando tocas un punto del mapa donde hay más de un profesional:

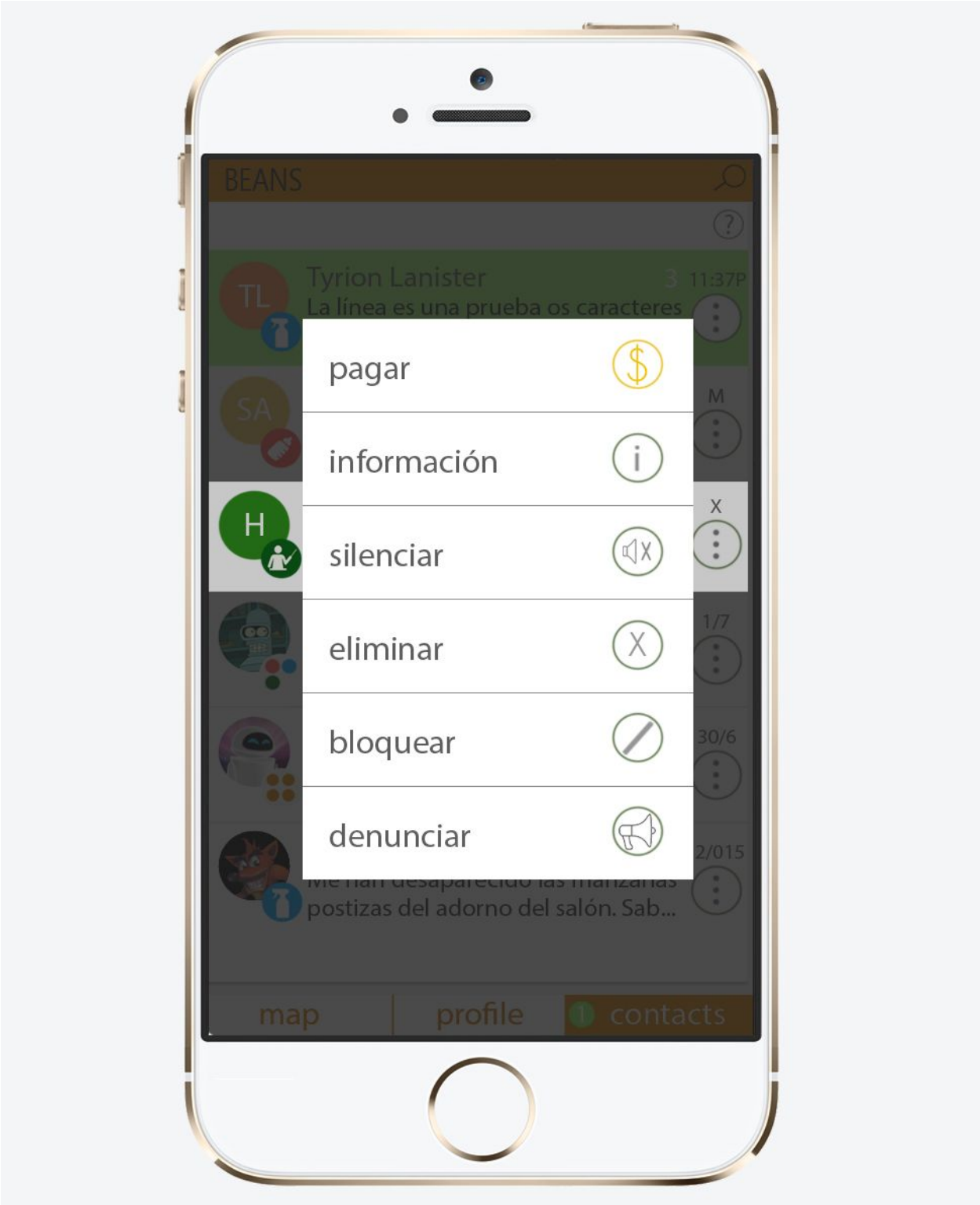
Lista de chats (en verde tarea pagada):



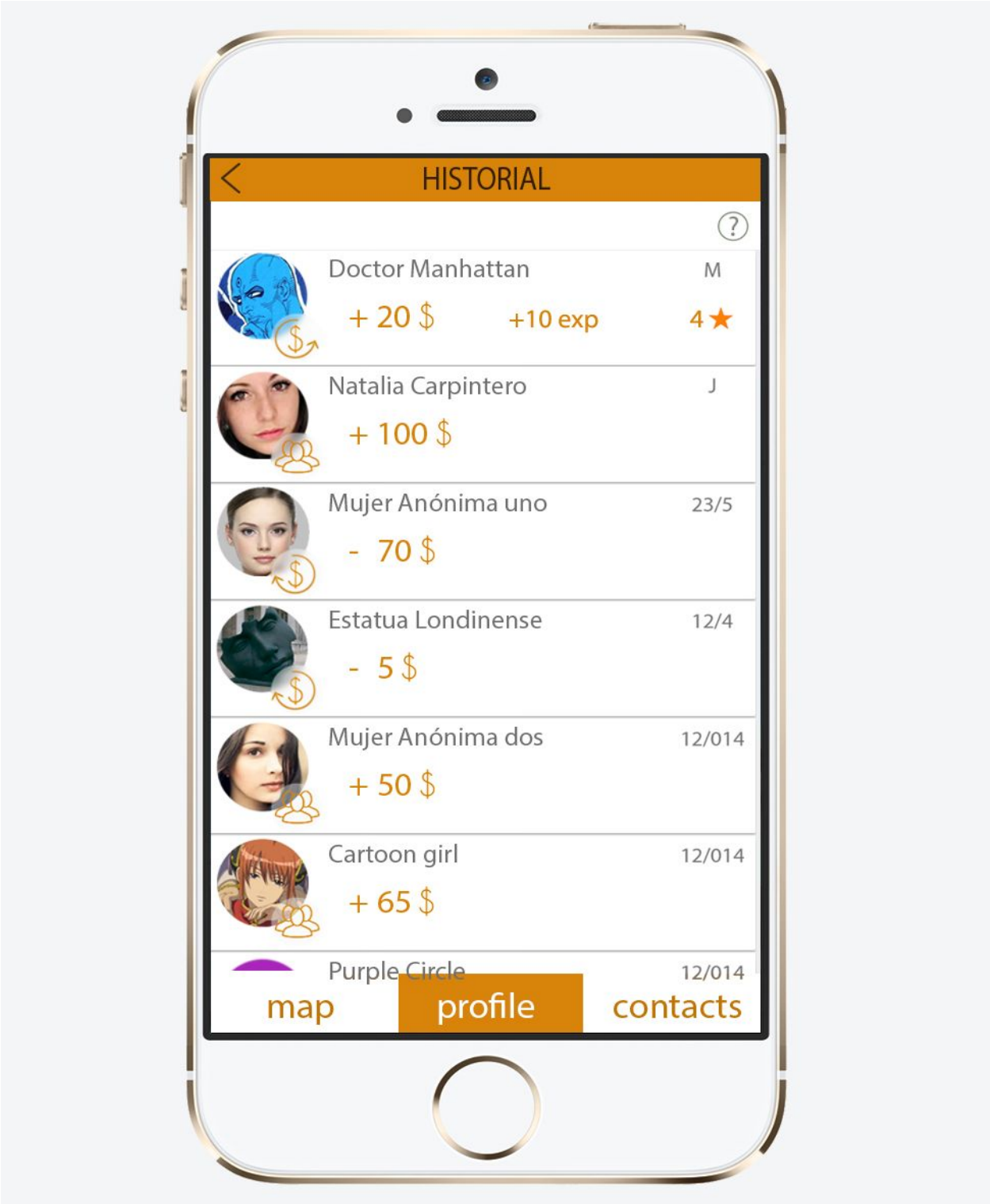
Lista de chats (en amarillo tarea realizada pero no pagada):



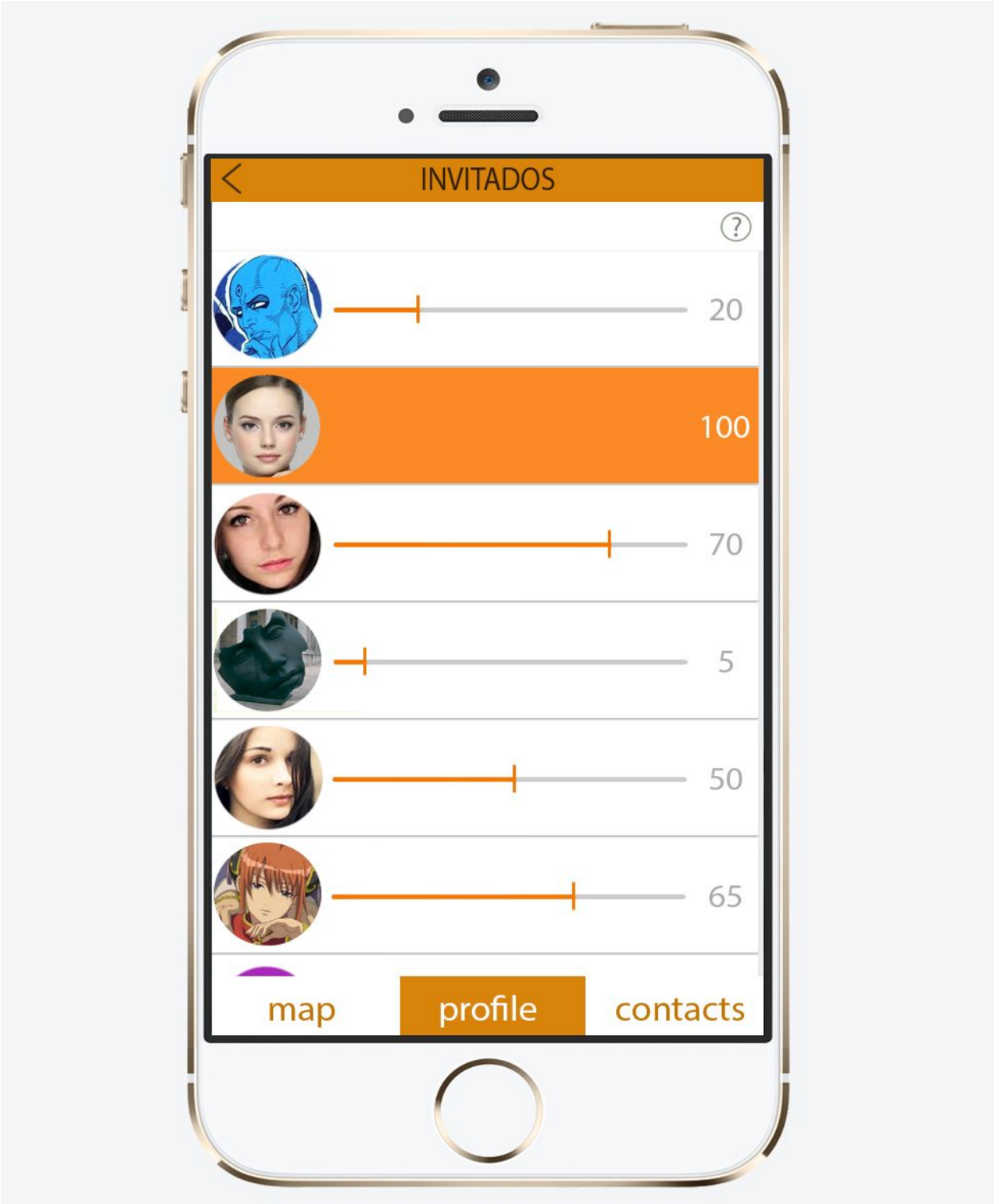
Acciones disponibles sobre un contacto del chat:



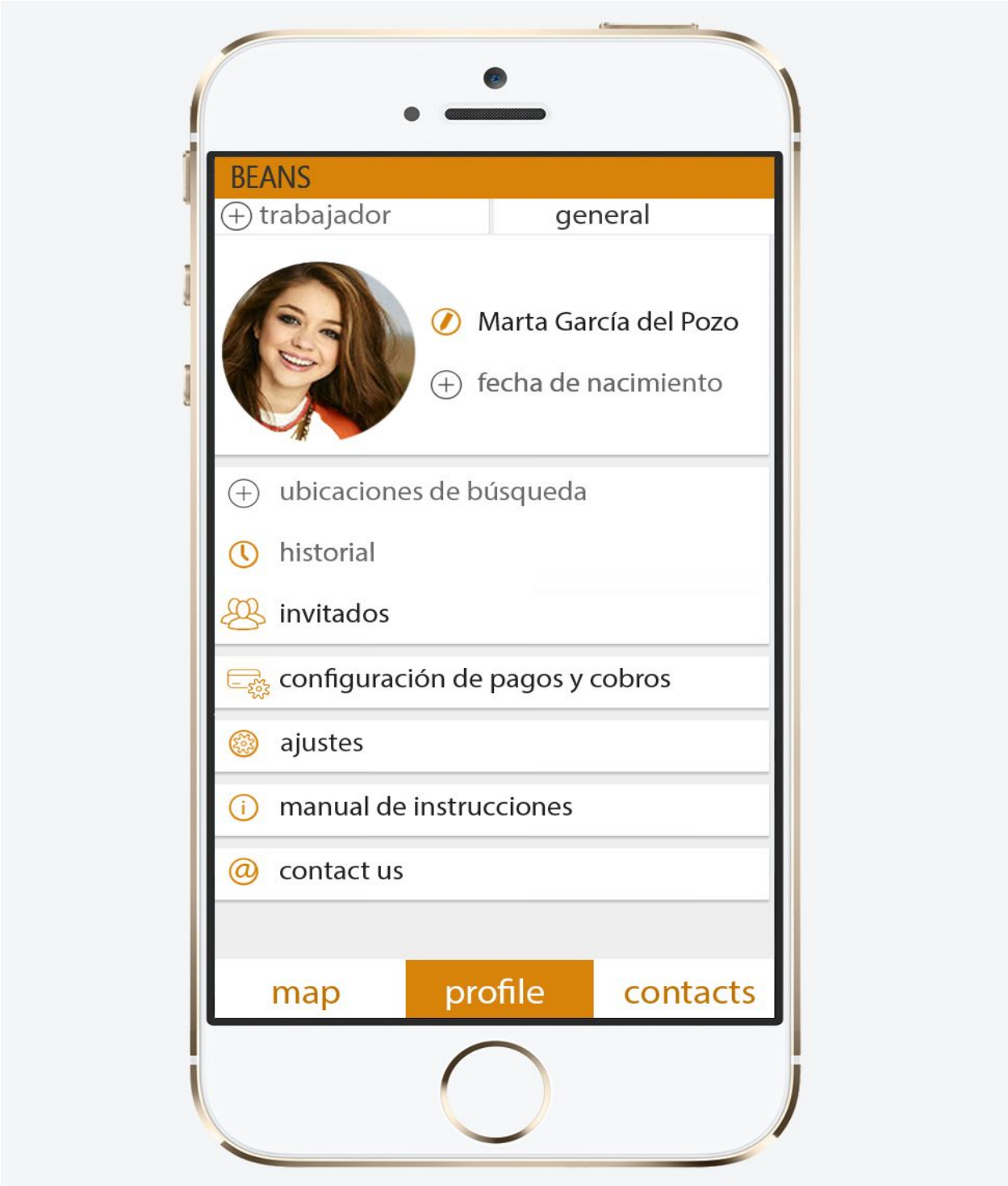
Listado de los propios cobros y pagos:



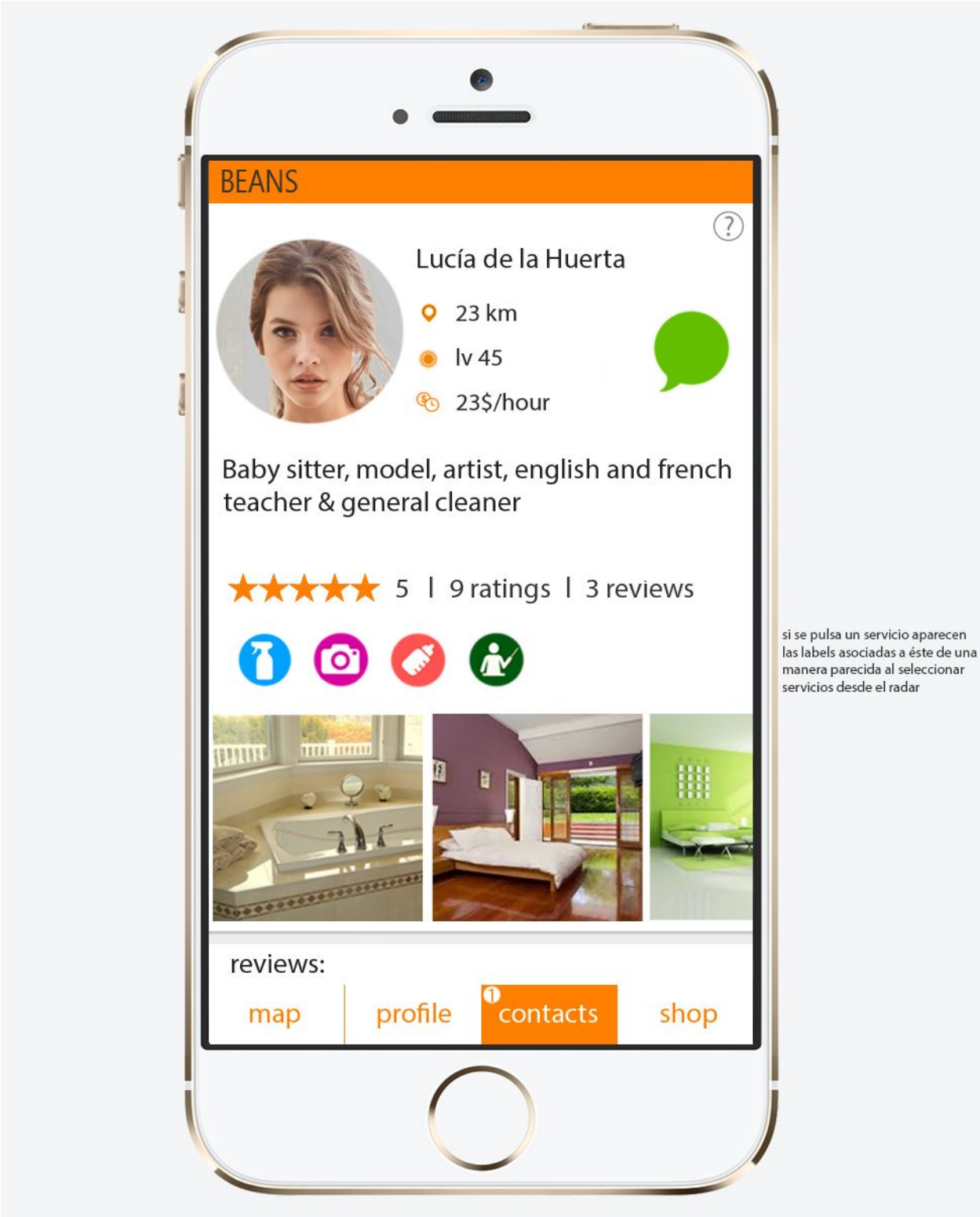
Listado del beneficio obtenido de cada invitado (hasta un máximo de 100€):



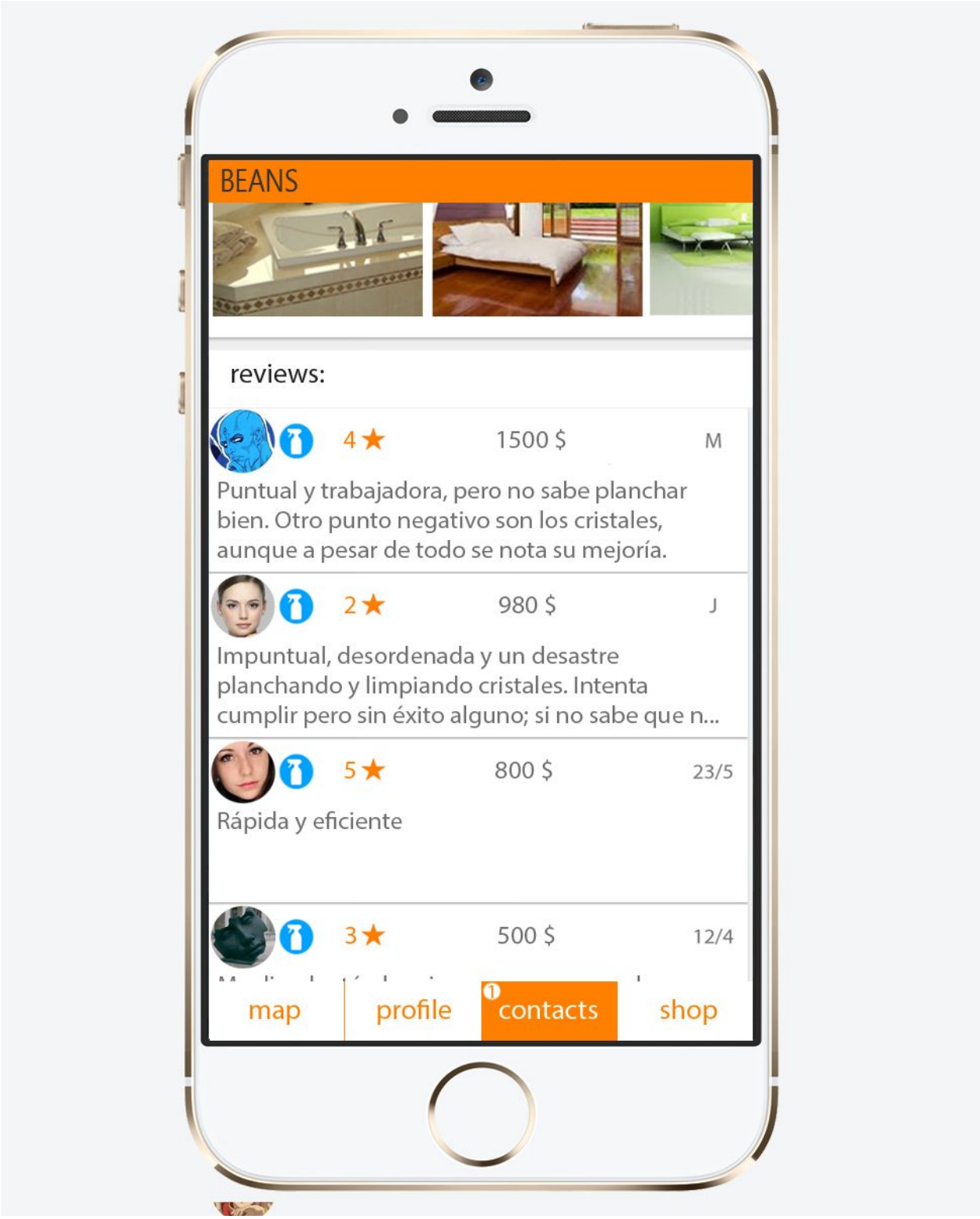
El propio perfil, desde donde se puede acceder al historial de cobros y pagos, y a la lista de invitados, ambos mostrados anteriormente:



Perfil de un profesional cercano que domina hasta cuatro profesiones:



Lista de valoraciones de un profesional:



8. Implementación

Este apartado se dividirá en tres secciones: en la primera, se detallará la implantación completa del sistema; en la segunda, se documentarán las APIs; y en la tercera, se explicará la interacción entre el cliente móvil y las APIs.

8.1. Implantación completa del sistema

Lo primero que hay que tener en cuenta es que los componentes tareas, pagos de tareas y valoraciones de tareas se han implementado finalmente como módulos de un mismo componente.

Esto nos deja con un total de cinco componentes:

1. Identidad y acceso.
2. Invitaciones.
3. Mensajería instantánea.
4. Geodirectorio de profesionales.
5. Tareas.

Todos ellos funcionan como archivos WAR ejecutándose dentro de un servidor de aplicaciones Apache Tomcat. Además, todos utilizan un servidor de bases de datos MySQL, excepto geodirectorio de profesionales, que utiliza un servidor PostgreSQL con extensión PostGIS.

Geodirectorio de profesionales se ha implantado sobre un Droplet Standard de Digital Ocean tipo V, con las siguientes especificaciones técnicas:

- Memoria: 8GB
- Procesador: 4 núcleos.
- Disco SSD: 80GB.
- Transferencia: 5TB.

Los demás, a saber, identidad y acceso, invitaciones, mensajería instantánea, y tareas, se han implantado cada uno sobre un Droplet Standard de Digital Ocean tipo II, con las siguientes especificaciones técnicas:

- Memoria: 2GB.
- Procesador: 2 núcleos.
- Disco SSD: 40GB.
- Transferencia: 3TB.

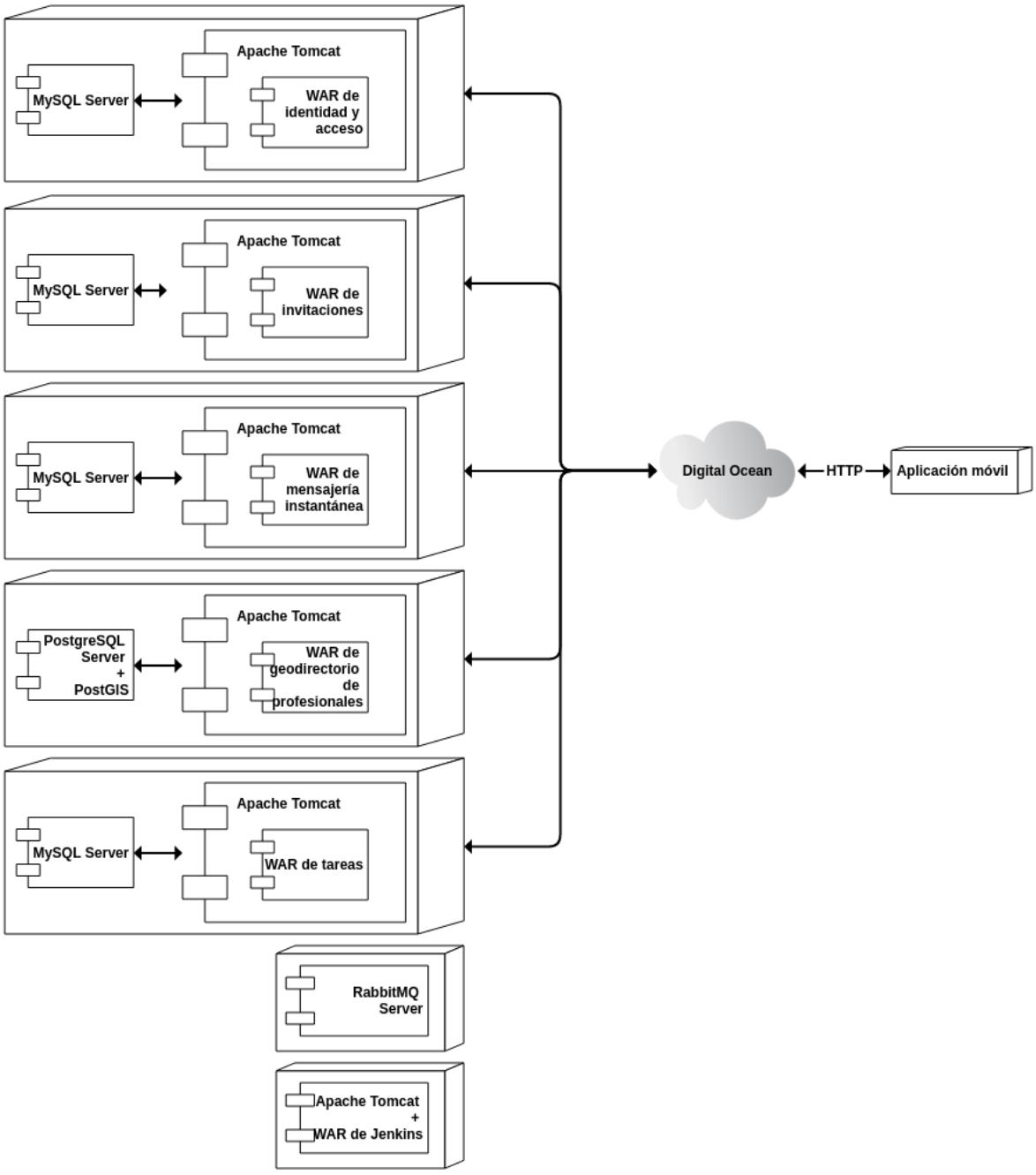
Además, disponemos de dos Droplet Standard tipo II adicionales: uno, que contiene el servidor de integración continua, un WAR de Jenkins servido por medio de un Apache Tomcat; el otro, que contiene el servidor de mensajería, un servidor RabbitMQ, que si bien se utilizó para probar diferentes versiones de los componentes en producción, ahora mismo no se usa.

Se ha optado por varios droplets de bajas especificaciones técnicas antes que por pocos de altas especificaciones técnicas por un motivo meramente económico: el precio no crece

linealmente con las especificaciones técnicas, sino que a medida que las especificaciones técnicas aumentan los precios se disparan.

Evidentemente, se contempla la posibilidad de ejecutar los cuatro componentes que no son geodirectorio de profesionales en el mismo droplet, utilizando la funcionalidad de anfitriones virtuales o virtual hosts de Apache Tomcat. De esta forma podríamos tener cuatro droplets sirviendo los cuatro componentes, y lo único que habría que hacer es implantar un balanceador de carga para que repartiera las peticiones entre droplets.

En definitiva, la implantación completa del sistema ha sido la siguiente:


Apache Tomcat
MySQL Server
WAR de identidad y acceso
Apache Tomcat
MySQL Server
WAR de invitaciones
Apache Tomcat
MySQL Server
WAR de mensajería instantánea
Apache Tomcat
PostgreSQL Server + PostGIS
WAR de geodirectorio de profesionales
Apache Tomcat
MySQL Server
WAR de tareas
RabbitMQ Server
Apache Tomcat + WAR de Jenkins
Digital Ocean
HTTP
Aplicación móvil

8.2. Los componentes de la aplicación: documentación de las APIs

En esta sección documentaré las partes principales de las APIs de los diferentes componentes.

8.2.1. Identidad y acceso

8.2.1.1. Registrar usuario

POST /users

Petición

- fullName (string, requerido): El nombre completo del usuario.
- emailAddress (string, requerido): La dirección de correo electrónico del usuario.
- password (string, requerido): La contraseña del usuario.
- userType (enum: {professional}, {customer}, requerido): El tipo de usuario.

Repuesta

- id (string): el identificador asignado al usuario.

8.2.1.2. Autenticar usuario

POST /users/authenticate

Petición

- emailAddress (string, requerido): La dirección de correo electrónico del usuario.
- password (string, requerido): La contraseña del usuario.

Repuesta

- userId (string): El identificador del usuario.
- userType (enum: {professional, customer}): El tipo de usuario del usuario.
- authenticationToken (string): El token de autenticación asignado al usuario.

8.2.1.3. Obtener usuarios

GET /users/{ids}

Parámetros URL

- {ids} (string): Los identificadores de los usuarios que se desea obtener. Los identificadores de usuario irán separados por comas.

Repuesta

- Users (array): Para cada usuario solicitado, se proporciona la siguiente información:
    - id (string): El identificador del usuario.
    - userType (string): El tipo de usuario del usuario.
    - fullName (string): El nombre completo del usuario.

- emailAddress (string): La dirección de correo electrónico del usuario.

8.2.1.4. Obtener usuario con cierto token de autenticación

GET /users/with-auth-token/{auth-token}

Parámetros URL

- auth-token (string): el token de autenticación del usuario.

Respuesta

- id (string): El identificador del usuario.
- userType (string): El tipo de usuario del usuario.
- fullName (string): El nombre completo del usuario.
- emailAddress (string): La dirección de correo electrónico del usuario.

8.2.2. Invitaciones

8.2.2.1. Registrar uso de invitación

POST /invitations

Petición

- inviterId (string, obligatorio): el identificador del invitador.
- invitedId (string, obligatorio): el identificador del invitado.

Repuesta

- id (string): el identificador asignado a la invitación usada.

8.2.2.2. Obtener invitaciones usadas de un invitador

GET /invitations/by-inviter-with-id/{inviter-id}

Parámetros URL

- {inviter-id} (string): El identificador del invitador.

Repuesta

- invitedId (string): El identificador del invitado.
- inviterId (string): El identificador del invitador.
- totalAmountRewardedToInviter (string): La cantidad con que se premió al invitador en derechos de participación en las tasas aplicadas a transacciones del invitado.
- totalRewardAmountClaimedByInviter (string): La cantidad a la que asciende el total de las participaciones del invitador en transacciones del invitado.

8.2.3. Mensajería instantánea

8.2.3.1. Colocar los datos del servicio de mensajería de un usuario

PUT /users/{id}

Parámetros URL:

- {id}: El identificador del usuario.

Petición

- remoteId: El identificador del usuario que utiliza Firebase Cloud Messaging para ubicar la instancia de la aplicación que está ejecutando dicho usuario y, por tanto, para ubicar el dispositivo de este usuario.

8.2.4. Geodirectorio de profesionales

8.2.4.1. Colocar los datos de un profesional

PUT /professionals/{id}

Parámetros URL

- {id}: El identificador del profesional.

Petición

- profession (enum: {carpenter, drywaller, roofer, painter, mason, electrician, plumber}): La profesión del profesional.
- streetName (string): El nombre completo de la vía donde el profesional tiene su centro de trabajo.
- blockNumber (integer): El número de bloque dentro de la vía donde el profesional tiene su centro de trabajo.

- postalCode (string): Código postal que le corresponde al centro de trabajo del profesional.
- cityName (string): Nombre de la ciudad donde está ubicado el centro del trabajo del profesional.
- countryName (string): Nombre del país donde está ubicado el centro de trabajo del profesional.
- latitudeInDegrees (double): Latitud del centro de trabajo del profesional.
- longitudeInDegrees (double): Longitud del centro de trabajo del profesional.

8.2.4.2. Obtener profesionales cercanos

GET

/professionals/closest-to-latitude/{latitude}/and-longitude/{longitude}/with-profession/{profession}

Parámetros URL

{latitude} (double): La latitud que especifica el punto en torno al cual queremos encontrar profesionales cercanos.

{longitude} (double): La longitud que especifica el punto en torno al cual queremos encontrar profesionales cercanos.

{profession} (enum: {carpenter, drywaller, roofer, painter, mason, electrician, plumber, any}: La profesión que queremos que desempeñen los profesionales cercanos.

page (integer, opcional): parámetro de consulta para especificar el número de la página de profesionales que deseamos obtener (por defecto, 1).

Respuesta

- professionals (array): Para cada profesional, se proporciona la siguiente información:
    - profession (enum: {carpenter, drywaller, roofer, painter, mason, electrician, plumber}): La profesión del profesional.
    - streetName (string): El nombre completo de la vía donde el profesional tiene su centro de trabajo.
    - blockNumber (integer): El número de bloque dentro de la vía donde el profesional tiene su centro de trabajo.
    - postalCode (string): Código postal que le corresponde al centro de trabajo del profesional.
    - cityName (string): Nombre de la ciudad donde está ubicado el centro del trabajo del profesional.
    - countryName (string): Nombre del país donde está ubicado el centro de trabajo del profesional.
    - latitudeInDegrees (double): Latitud del centro de trabajo del profesional.
    - longitudeInDegrees (double): Longitud del centro de trabajo del profesional.

8.2.5. Tareas

8.2.5.1. Crear tarea

POST /jobs

Petición

- jobRequesterId (string, obligatorio): El identificador del solicitante de la tarea.
- jobDoerId (string, obligatorio): El identificador del realizador de la tarea.
- jobDescription (string, obligatorio): La descripción de la tarea.
- jobPrice (string, obligatorio): El precio de la tarea.

Repuesta

- {id}: El identificador asignado a la tarea.

8.2.5.2. Pagar tarea

POST /jobs/{id}/pay

Parámetros URL

- {id}: El identificador de la tarea que se va a pagar.

Cabecera de la petición

- X-Token (string, obligatorio): el token de autenticación del usuario que realiza la petición. Nótese que solamente el solicitante de la tarea puede proceder al pago.

Respuesta

- id: El identificador asignado al pago.
- remoteId: El identificador del pago en PayPal, a partir del cual se puede proceder a la confirmación del pago.

8.2.5.3. Valorar tarea

POST /jobs/{id}/review

Parámetros URL

- {id}: El identificador de la tarea que se valora.

Cabecera de la petición

- X-Token (string, obligatorio): el token de autenticación del usuario que realiza la petición. Nótese que solamente el solicitante de la tarea puede valorarla.

Petición

- reviewTitle (string, obligatorio): título de la valoración.
- reviewBody (string, obligatorio): cuerpo de la valoración.
- ratingInStars (double, obligatorio): la puntuación en estrellas.

Respuesta

- id: El identificador asignado a la valoración.

8.2.5.4. Colocar los datos del servicio de pago de un usuario

PUT /professionals/{id}

Parámetros URL

- {id}: El identificador del profesional.

Petición

- {remoteId}: El identificador del profesional en PayPal.

8.2.5.5. Obtener tareas

GET /jobs/{ids} (para obtener una selección de tareas)

o

GET /jobs/requested-by/{customer-id} (para obtener tareas solicitadas por un cliente)

o

GET /jobs/requested-to/{professional-id} (para obtener tareas solicitadas a un profesional)

Parámetros de las URLs

- {ids} (string): los identificadores de las tareas que se desea obtener.
- {customer-id} (string): el identificador del cliente.
- {professional-id} (string): el identificador del profesional.

Respuesta

- jobs (array): Para cada tarea, se proporciona la siguiente información:
    - id (string): El identificador de la tarea.
    - requesterId (string): El identificador del solicitante de la tarea.
    - doerId (string): El identificador de aquel a quien se ha solicitado la realización de la tarea.
    - description (string): La descripción de la tarea.
    - price (string): El precio de la tarea.
    - createdOn (string): La fecha de creación de la tarea.
    - status (enum: {created, paid, reviewed}): El estado actual de la tarea.

8.2.5.6. Obtener pagos

GET /job-payments/by-payer/{payer-id} (para obtener pagos realizados por un cliente)

o

GET /job-payments/to-payee/{payee-id} (para obtener los pagos realizados a un profesional)

o

GET /job-payments/of-jobs/{job-ids} (para obtener los pagos correspondientes a una o varias tareas)

Parámetros de las URLs

- {payer-id} (string): el identificador del pagador.

- {payee-id} (string): el identificador del pagado.
- {job-ids} (string): los identificadores de las tareas.

Respuesta

- jobPayments (array): Para cada pago, se proporciona la siguiente información:
    - id (string): El identificador del pago.
    - remoteId (string): El identificador del pago en PayPal.
    - jobId (string): El identificador de la tarea.
    - payerId (string): El identificador del pagador.
    - payeeId (string): El identificador del pagado.
    - paymentAmount (string): La cantidad pagada.
    - status (enum: {prepared, authorized, cancelled, appOwnerPaid, professionalPaid}): El estado actual del pago.
    - authorizedOn (string): La fecha de autorización del pago.

8.2.5.7. Obtener valoraciones

GET /reviews/by-reviewer/{reviewer-id} (para obtener las valoraciones realizadas por un cliente)

o

GET /reviews/of-jobs-by/{professional-id} (para obtener las valoraciones correspondientes a tareas realizadas por un profesional)

o

GET /reviews/of-jobs/{job-ids} (para obtener las valoraciones correspondientes a una o varias tareas)

Parámetros de las URLs

- {reviewer-id} (string): el identificador del valorador.
- {professional-id} (string): el identificador del profesional.
- {job-ids} (string): los identificadores de las tareas.

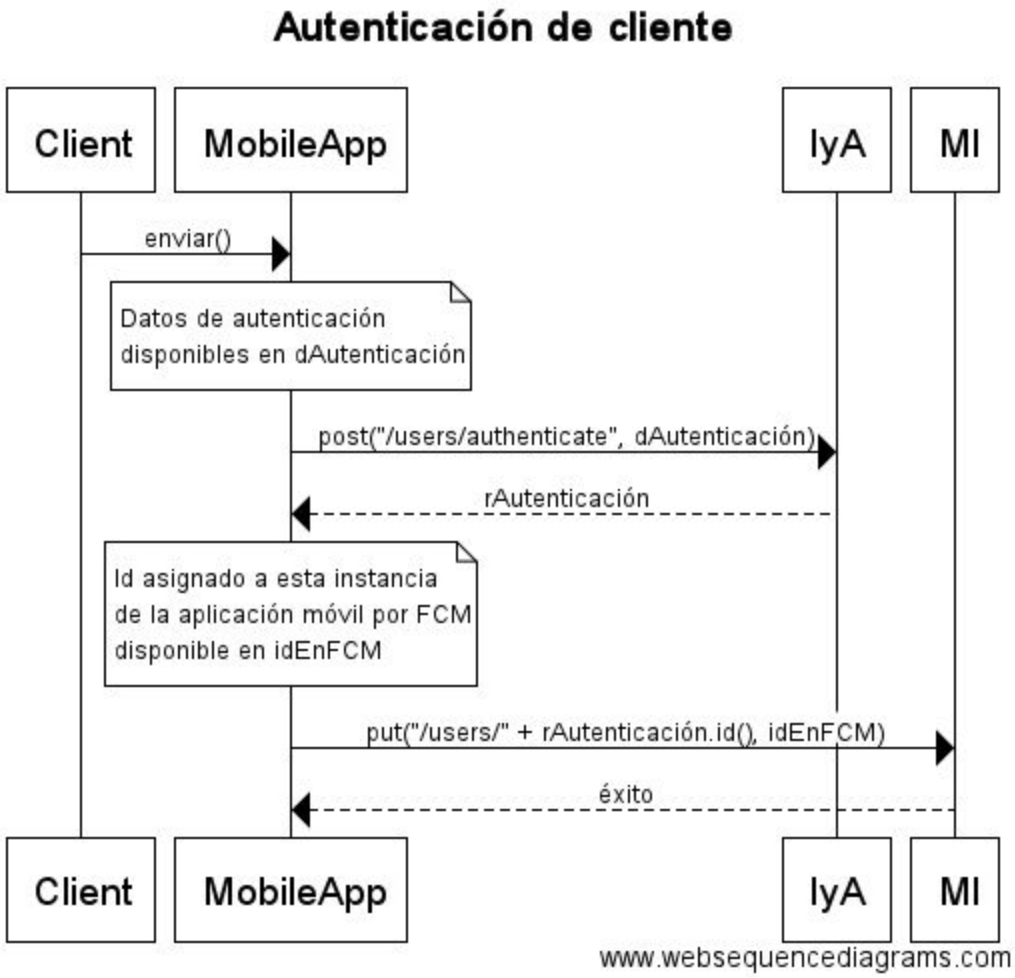
Respuesta

- reviews (array): Para cada valoración, se proporciona la siguiente información:
    - id (string): El identificador de la valoración.
    - reviewerId (string): El identificador del valorador.
    - jobId (string): El identificador de la tarea cuya realización se valora.
    - professionalId (string): El identificador del profesional que ha realizado la tarea.
    - reviewTitle (string): El título de la valoración.
    - reviewBody (string): El cuerpo de la valoración.
    - ratingInStars (double): La puntuación en estrellas.
    - publishedOn (string): La fecha de publicación.

8.3. El cliente móvil: integración de APIs

En esta sección muestro y explico cómo he implementado la integración de APIs en el cliente móvil, y para ello me serviré de una selección de casos de uso que considero representativos.

8.3.1. Autenticación de un cliente

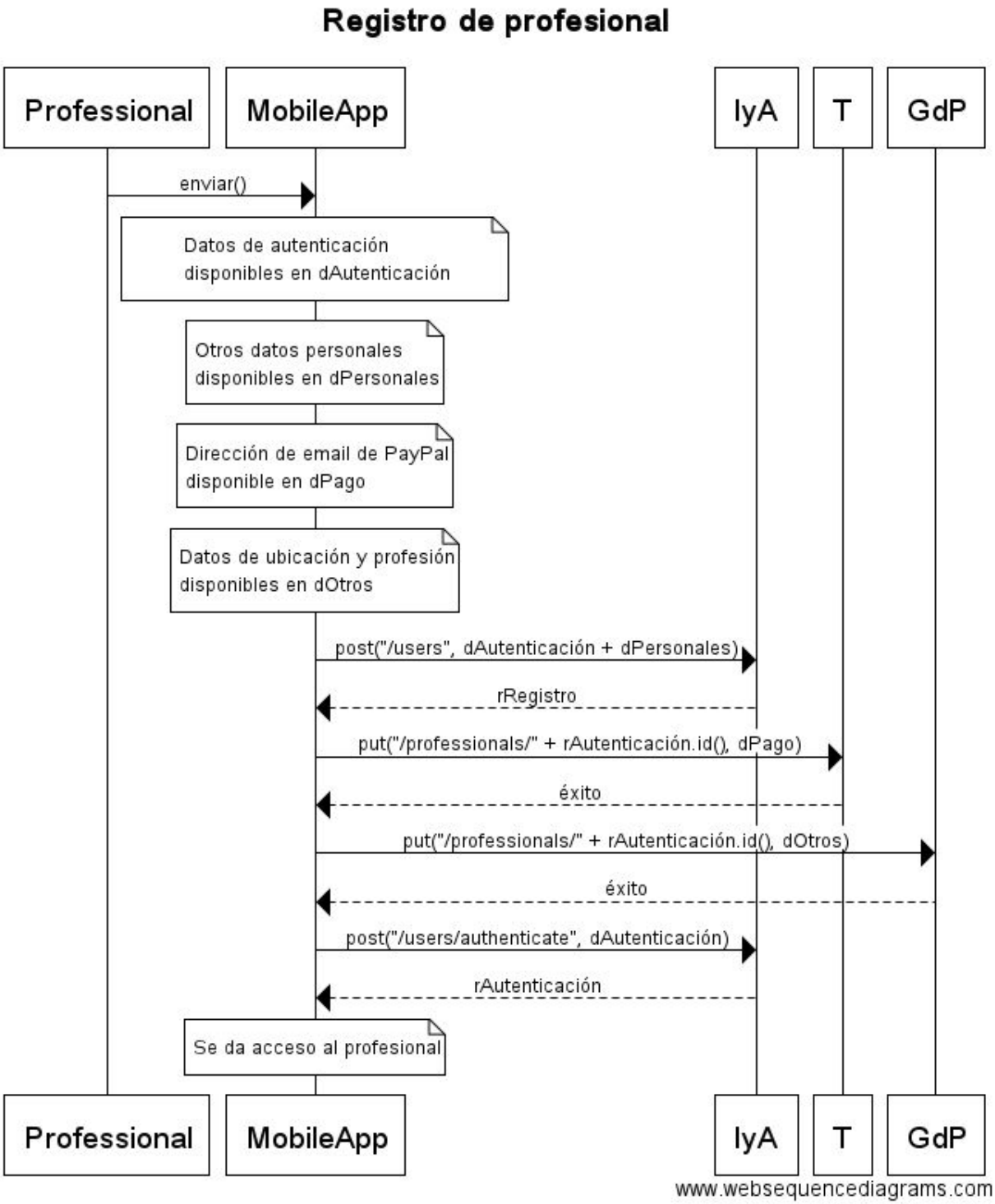


Básicamente, tras autenticar al usuario, la aplicación móvil tiene que actualizar el identificador de la instancia móvil a la que deben enviarse los mensajes, puesto que queremos recibirlos en este dispositivo.

Si se quiere obtener más tarde estos mensajes en otro dispositivo, se pueden consultar los mensajes entregados que se encuentran almacenados en el componente de mensajería instantánea. Para evitar tener que consultar y procesar todos los mensajes, lo que se podría hacer es especificar el identificador del último mensaje de que se dispone en el dispositivo actual, de modo que solamente se recibirían mensajes posteriores a ese.

IyA representa el componente de identidad y acceso, mientras que MI representa el componente de mensajería instantánea.

8.3.2. Registro de un profesional

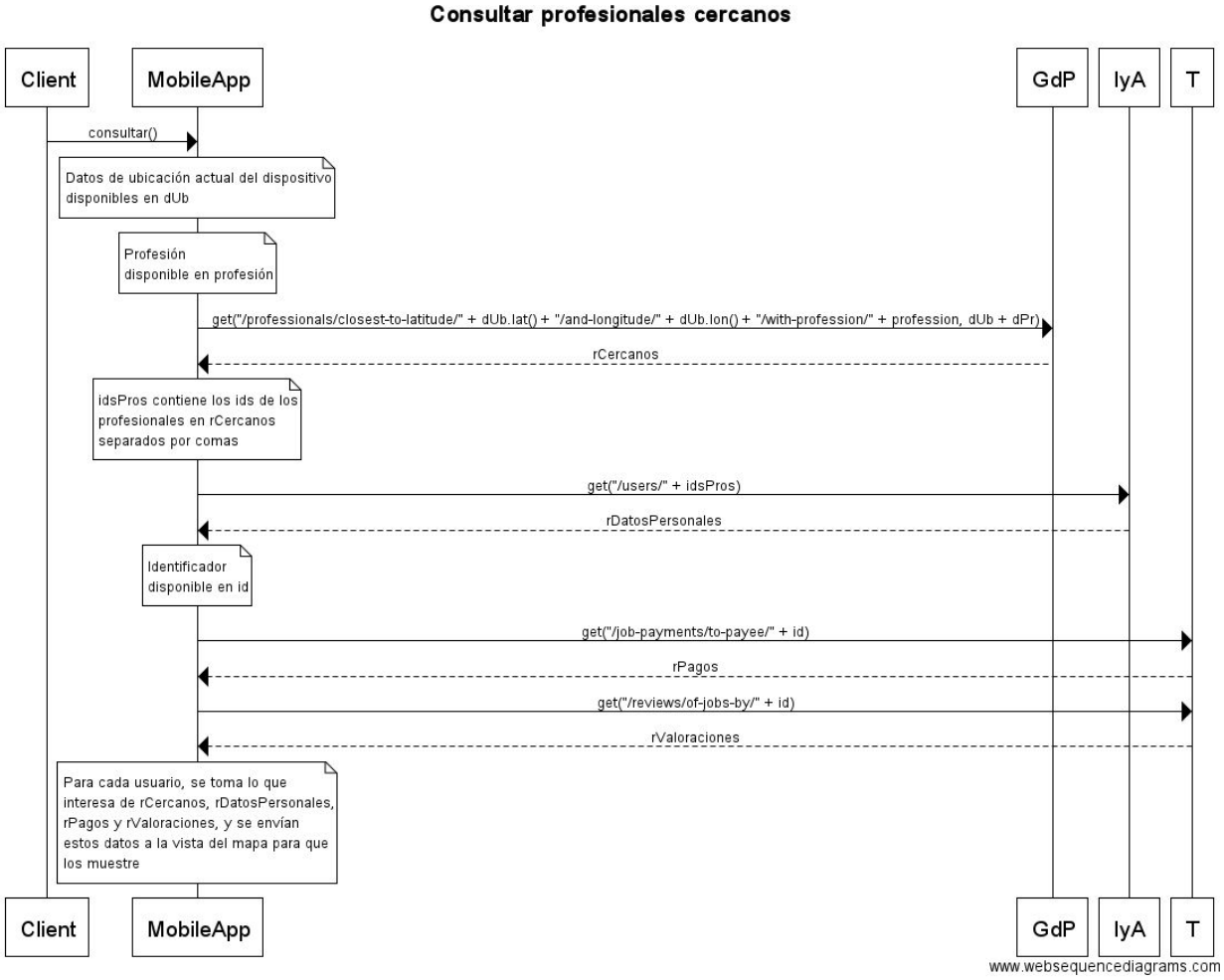


En este caso, lo que tenemos que hacer es distribuir la información que proporciona al registrarse el profesional entre tres componentes diferentes: identidad y acceso, tareas, y geodirectorio de profesionales.

Además, dado que no se requiere actualmente verificación de la dirección de correo electrónico proporcionada en el registro, lo que hacemos es autenticar directamente al profesional y darle acceso a la aplicación tras su registro.

T representa el componente de tareas, mientras que GdP representa el componente de geodirectorio de profesionales.

8.3.3. Obtener profesionales cercanos

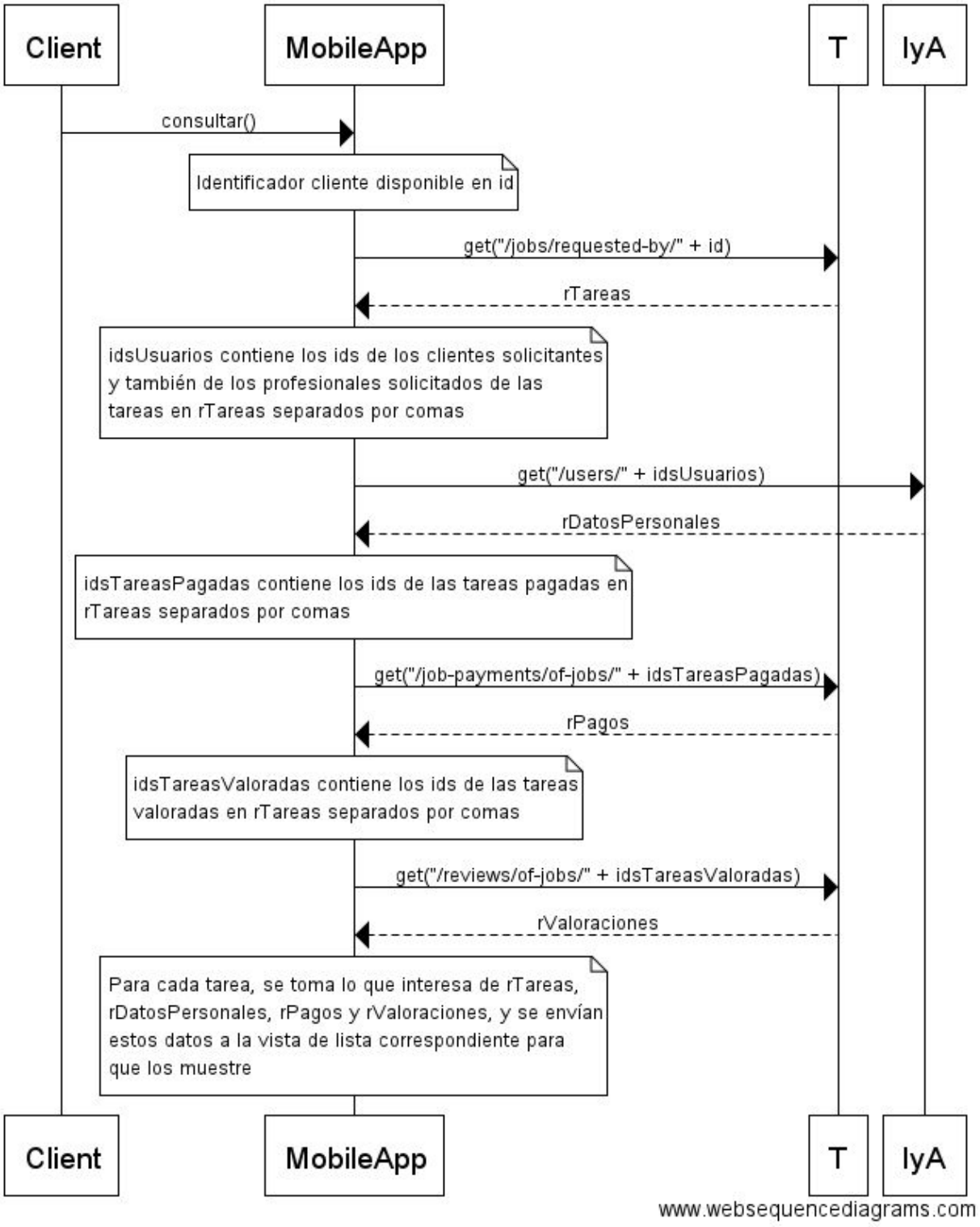


Una vez obtenemos los datos de ubicación y la profesión de los profesionales más cercanos, necesitamos obtener sus datos personales, sus pagos (para poder mostrar el total de ingresos) y sus valoraciones (para poder mostrar su valoración media).

Una vez tenemos toda la información necesaria, cogemos solamente lo que nos interesa y agrupamos la información por profesional. El resultado de este proceso ya se puede enviar a una vista de mapa o a una vista de lista para ser mostrado.

8.3.4. Obtener tareas solicitadas por cliente

Obtener tareas solicitadas por cliente

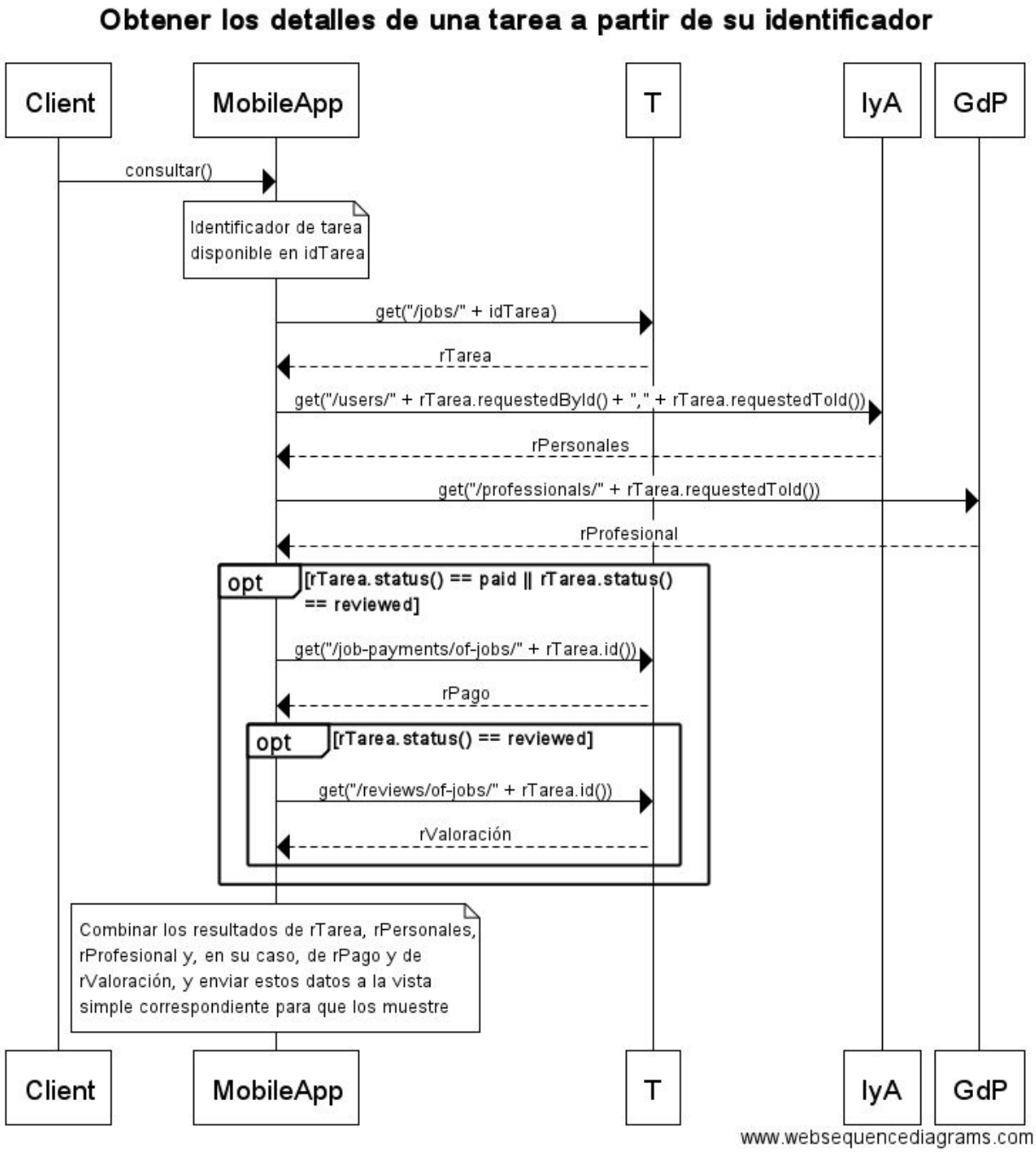


Una vez tenemos los datos de las tareas, necesitamos saber el nombre tanto del cliente solicitante como del profesional solicitado, además de información adicional de pago en aquellas tareas que hayan sido pagadas, y de información adicional de valoración en aquellas tareas que hayan sido valoradas.

Con toda la información necesaria, ya podemos coger lo que nos interesa y agrupar la información por tarea. El resultado de este proceso se puede pasar directamente a la vista de lista encargada de mostrarlo.

El caso de uso correspondiente a la obtención por parte de un profesional de las tareas que le han sido solicitadas resultaría similar.

8.3.5. Obtener los detalles de una tarea a partir de su identificador

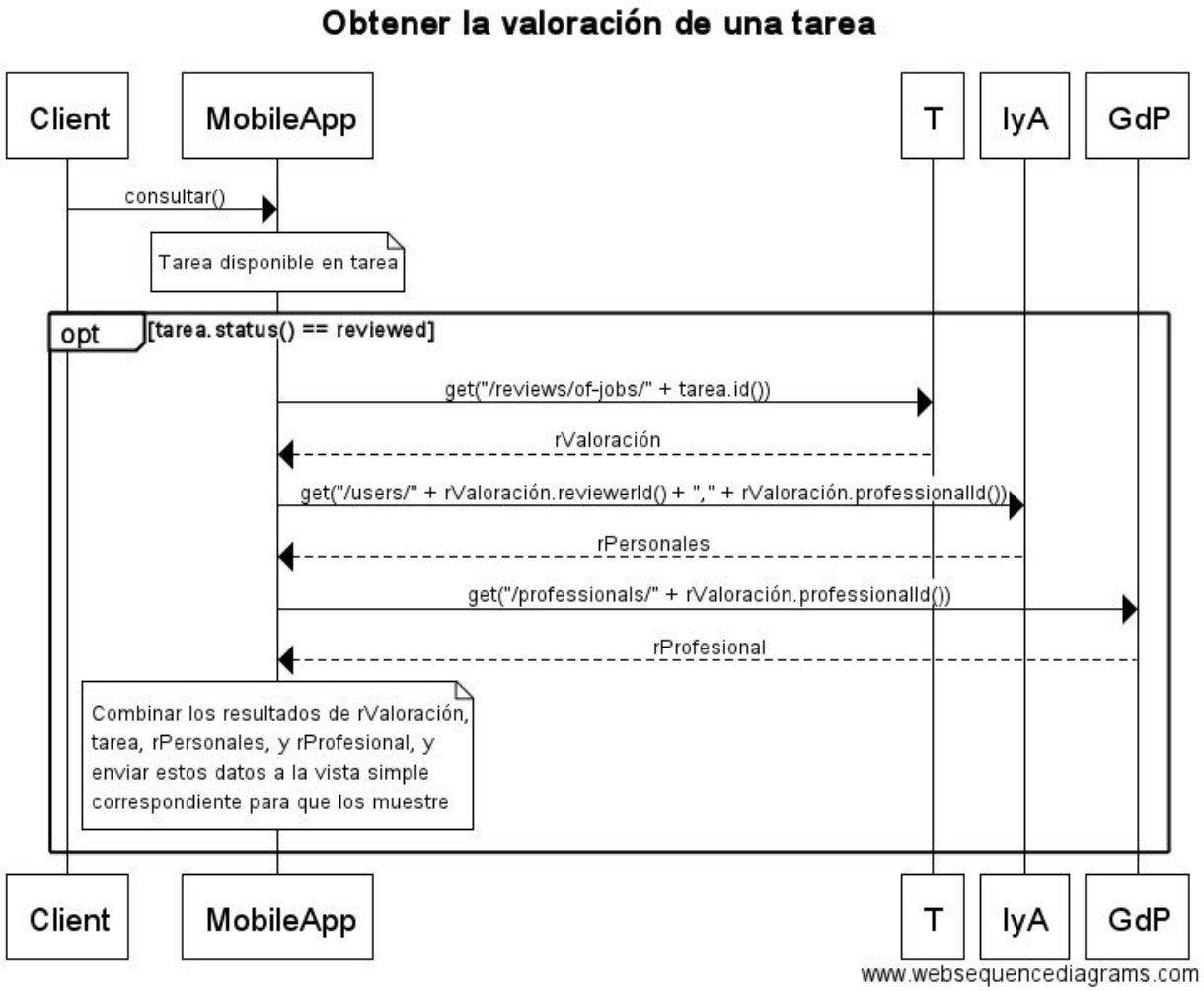


Cuando obtenemos una tarea a partir de su identificador, lo siguiente será obtener el nombre tanto del solicitante de la tarea como del profesional a quien se ha solicitado, la profesión del

profesional, los detalles del pago si la tarea ha sido pagada, y los detalles de la valoración si la tarea ha sido valorada.

Una vez disponemos de toda la información deseada, es cuestión de quedarnos con la parte de la información que nos interesa, organizarla, y pasársela a una vista simple para que la muestre.

8.3.6. Obtener la valoración de una tarea



A partir de la tarea, podemos obtener su valoración, información que después complementaremos con el nombre tanto del valorador como del valorado, y con la profesión del valorado.

Una vez organizada, dicha información puede ser enviada a la vista simple correspondiente para su muestra.

9. Pruebas

Este apartado consta de dos secciones: por un lado, de la especificación de las pruebas de aceptación; por otro, de las pruebas efectivamente realizadas.

9.1. Especificación de las pruebas de aceptación

En esta sección, seguiré aplicando desarrollo guiado por el comportamiento para definir las pruebas de aceptación del sistema. Partiré de la lista de requerimientos funcionales; dividiré aquellos que abarquen demasiado en historias de usuario, e ilustraré los requerimientos funcionales no divididos y las historias de usuario con ejemplos.

Estos ejemplos se pueden formalizar, en cuyo caso pasan a ser especificaciones de pruebas de aceptación. Obtener dichas especificaciones, que luego deben ser implementadas, es el objetivo de esta sección.

El formato propuesto por quienes practican BDD para especificar las pruebas de aceptación es el siguiente:

Dado <un contexto>

Cuando <ocurre algo>

Entonces <esperamos algún resultado>

Dicho esto, los requerimientos funcionales para los cuales especificaré las pruebas de aceptación son los siguientes:

9.1.1. Permitir a los usuarios registrarse

Este requerimiento funcional lo dividiré en dos historias de usuario.

9.1.1.1. Registro con dirección de correo electrónico válida

Dado que María quiere registrarse con dirección de correo electrónico “maria@usuario.com”

Y que no hay ningún usuario registrado con dirección de correo electrónico “maria@usuario.com”

Cuando María envía la solicitud para registrarse

Entonces la solicitud debería ser aceptada

Y a María se le debería informar de que su solicitud ha sido aceptada

9.1.1.2. Registro con dirección de correo no válida

Dado que María quiere registrarse con dirección de correo electrónico “maria@usuario.com”

Y que ya hay un usuario registrado con dirección de correo electrónico “maria@usuario.com”

Cuando María envía la solicitud para registrarse

Entonces la solicitud debería ser rechazada

Y a María se le debería informar de que su dirección de correo electrónico ya está siendo utilizada por otro usuario

9.1.2. Permitir a los usuarios autenticarse

Este requerimiento funcional lo dividiré en dos historias de usuario.

9.1.2.1. Autenticación con credenciales válidas

Dado que María se registró con contraseña “secreto”

Cuando María se autentica con contraseña “secreto”

Entonces a María se le debería dar acceso a la aplicación

9.1.2.2. Autenticación con credenciales inválidas

Dado que María se registró con contraseña “secreto”

Cuando María se autentica con contraseña “error”

Entonces a María se le debería informar de que la contraseña proporcionada es incorrecta

9.1.3. Permitir a los clientes obtener un informe con los datos personales de un profesional

Dado que José es un cliente

Y que Antonio es un profesional

Cuando José consulta los datos personales de Antonio

Entonces José debería ver los datos personales de Antonio

9.1.4. Permitir a los profesionales obtener un informe con los datos personales de un cliente

Dado que Antonio es un profesional

Y que José es un cliente

Cuando Antonio consulta los datos personales de José

Entonces Antonio debería ver los datos personales de José

9.1.5. Permitir al cliente iniciar un chat con un profesional

Dado que María es una cliente

Y que Carmen es una profesional

Cuando María abre un chat con Carmen

Entonces el chat debería estar visible en la lista de chats de María

9.1.6. Permitir al cliente el envío de mensajes a un profesional. Envío de mensajes a un profesional que le han sido enviados por un cliente

Dado que José es un cliente

Y que Antonio es un profesional

Y que José ha escrito un mensaje en el chat que mantiene con Antonio

Cuando José envía el mensaje

Entonces el mensaje debería estar visible como mensaje enviado en el chat que José mantiene con Antonio

Y Antonio debería ser notificado del mensaje que le ha enviado José

Y el mensaje debería estar visible como mensaje recibido en el chat que Antonio mantiene con José

9.1.7. Permitir al profesional el envío de mensajes a un cliente. Envío de mensajes a un cliente que le han sido enviados por un profesional

Dado que Antonio es un profesional

Y que José es un cliente

Y que Antonio ha escrito un mensaje en el chat que mantiene con José

Cuando Antonio envía el mensaje

Entonces el mensaje debería estar visible como mensaje enviado en el chat que Antonio mantiene con José

Y José debería ser notificado del mensaje que le ha enviado Antonio

Y el mensaje debería estar visible como mensaje recibido en el chat que José mantiene con Antonio

9.1.8. Permitir a los usuarios eliminar un chat.

Dado que María quiere eliminar su chat con Carmen

Cuando María elimina su chat con Carmen

Entonces el chat ya no debería estar visible en la lista de chats de María

9.1.9. Permitir al profesional especificar sus datos de ubicación

Dado que Antonio es un profesional

Y que Antonio ha completado sus datos de ubicación

Cuando Antonio envía estos datos de ubicación

Entonces estos datos de ubicación deberían ser visibles en el perfil de Antonio

9.1.10. Permitir al profesional actualizar sus datos de ubicación

Dado que Antonio es un profesional

Y que Antonio ha completado sus nuevos datos de ubicación

Cuando Antonio envía estos nuevos datos de ubicación

Entonces estos nuevos datos de ubicación deberían ser visibles en el perfil de Antonio

9.1.11. Permitir al profesional especificar su profesión

Dado que Carmen es una profesional

Y que Carmen ha seleccionado su profesión

Cuando Carmen envía su profesión

Entonces esta profesión debería ser visible en el perfil de Carmen

9.1.12. Permitir al cliente obtener un informe con profesionales cercanos

Dado que María es una cliente

Cuando María consulta los profesionales cercanos

Entonces María debería ver una lista de profesionales ordenada ascendentemente por la distancia que separa a María del profesional

Un ejemplo más concreto

Esta especificación de prueba de aceptación requiere de un ejemplo más concreto, que es el siguiente:

Dado que María es una cliente

Y que Manuel es un profesional

Y que Manuel se encuentra a 100 metros de María

Y que Francisco es un profesional

Y que Francisco se encuentra a 250 metros de María

Y que Josefa es una profesional

Y que Josefa se encuentra a 500 metros de María

Cuando María consulta los profesionales cercanos

Entonces Manuel debería aparecer en la lista antes que Francisco

Y Francisco debería aparecer en la lista antes que Josefa

9.1.13. Permitir al cliente obtener un informe con profesionales cercanos de determinada profesión

Dado que María es una cliente

Y que María ha seleccionado una profesión

Cuando María consulta los profesionales cercanos

Entonces María debería ver una lista de profesionales cercanos de la profesión seleccionada ordenada ascendentemente por la distancia que separa a María del profesional

Un ejemplo más concreto

Esta especificación de prueba de aceptación requiere de un ejemplo más concreto, que es el siguiente:

Dado que María es una cliente

Y que María ha seleccionado la profesión “albañil”

Y que Manuel es un profesional

Y que Manuel se encuentra a 100 metros de María

Y que Manuel es un albañil

Y que Francisco es un profesional

Y que Francisco se encuentra a 250 metros de María

Y que Francisco es un electricista

Y que Josefa es una profesional

Y que Josefa se encuentra a 500 metros de María

Y que Josefa es una albañil

Y que Juan es un profesional

Y que Juan se encuentra a 750 metros de María

Y que Juan es un albañil

Cuando María consulta los profesionales cercanos

Entonces Manuel debería aparecer en la lista antes que Josefa

Y Josefa debería aparecer en la lista antes que Juan

Pero Francisco no debería aparecer en la lista

9.1.14. Permitir al profesional especificar la cuenta en la que desea recibir sus pagos

Dado que Antonio es un profesional

Y que Antonio ha completado la cuenta en la que desea recibir sus pagos

Cuando Antonio envía esta cuenta

Entonces esta cuenta debería ser visible para Antonio en su perfil

9.1.15. Permitir al cliente obtener un informe con el total cobrado por un profesional a través de la aplicación

Dado que María es una cliente

Y que Carmen es una profesional

Cuando María consulta el total cobrado por Carmen a través de la aplicación

Entonces María debería ver ese total cobrado por Carmen a través de la aplicación

9.1.16. Permitir al cliente pagar al profesional que le ha realizado una tarea

Este requerimiento funcional lo dividiremos en dos historias de usuario.

9.1.16.1. Pago con credenciales de pago válidas

Dado que José es un cliente

Y que Antonio es un profesional

Y que Antonio le ha realizado una tarea a José

Y que José ha indicado que quiere pagar dicha tarea

Y que José ha introducido unas credenciales de pago válidas cuando se le han requerido

Cuando José confirma el pago

Entonces a José se le debería informar de que el pago se ha procesado correctamente

Y la tarea debería estar visible como pagada en la lista de tareas solicitadas por José

Y la tarea debería estar visible como pagada en la lista de tareas solicitadas a Antonio

9.1.16.2. Pago con credenciales de pago inválidas

Dado que José es un cliente

Y que Antonio es un profesional

Y que Antonio le ha realizado una tarea a José

Y que José ha indicado que quiere pagar dicha tarea

Pero que José ha introducido unas credenciales de pago inválidas cuando se le han requerido

Cuando José confirma el pago

Entonces a José se le debería informar de que las credenciales de pago que ha introducido son inválidas

9.1.17. Permitir a los clientes solicitar una tarea a un profesional

Dado que María es una clienta

Y que Carmen es una profesional

Y que María ha completado una solicitud de tarea dirigida a Carmen

Cuando María envía la solicitud

Entonces a María se le debería informar de que su solicitud se ha procesado correctamente

Y la tarea debería estar visible como solicitada en la lista de tareas solicitadas por María

Y la tarea debería estar visible como solicitada en la lista de tareas solicitadas a Carmen

9.1.18. Permitir a los profesionales realizar un presupuesto para una tarea solicitada por un cliente

Dado que Carmen es una profesional

Y que María es una clienta

Y que María solicitó una tarea a Carmen

Y que Carmen ha realizado un presupuesto para la tarea que le solicitó María

Cuando Carmen envía el presupuesto

Entonces a Carmen se le debería informar de que su presupuesto se ha procesado correctamente

Y la tarea debería estar visible como presupuestada en la lista de tareas solicitadas por María

Y la tarea debería estar visible como presupuestada en la lista de tareas solicitadas a Carmen

9.1.19. Permitir a los clientes rechazar un presupuesto realizado por un profesional

Dado que María es una clienta

Y que Carmen es una profesional

Y que María solicitó una tarea a Carmen

Y que Carmen presupuestó la tarea que le solicitó María

Y que María está viendo el presupuesto que le hizo Carmen

Cuando María rechaza el presupuesto

Entonces la tarea debería estar visible como presupuestada pero rechazada en la lista de tareas solicitadas por María

Y la tarea debería estar visible como presupuestada pero rechazada en la lista de tareas solicitadas a Carmen

9.1.20. Permitir a los clientes aceptar un presupuesto realizado por un profesional

Dado que María es una clienta

Y que Carmen es una profesional

Y que María solicitó una tarea a Carmen

Y que Carmen presupuestó la tarea que le solicitó María

Y que María está viendo el presupuesto que le hizo Carmen

Cuando María acepta el presupuesto

Entonces la tarea debería estar visible como presupuestada y aceptada en la lista de tareas solicitadas por María

Y la tarea debería estar visible como presupuestada y aceptada en la lista de tareas solicitadas a Carmen

9.1.21. Permitir al cliente valorar el desempeño del profesional en la realización de determinada tarea

Dado que María es una clienta

Y que Carmen es una profesional

Y que María solicitó una tarea a Carmen

Y que Carmen presupuestó la tarea que le solicitó María

Y que María aceptó el presupuesto

Y que Carmen realizó la tarea

Y que María pagó la tarea

Y que María ha completado la valoración de la tarea

Cuando María envía la valoración

Entonces María debería ser informada de que su valoración se ha procesado correctamente

Y la tarea debería estar visible como valorada en la lista de tareas solicitadas por María

Y la tarea debería estar visible como valorada en la lista de tareas solicitadas a Carmen

Y la valoración debería estar visible en la lista de valoraciones realizadas por María

Y la valoración debería estar visible en la lista de valoraciones realizadas a Carmen

9.1.22. Permitir al cliente obtener un informe con las tareas que ha solicitado

Dado que José es un cliente

Cuando José consultas las tareas que ha solicitado

Entonces José debería ver las tareas que ha solicitado

9.1.23. Permitir al profesional obtener un informe con las tareas que se le han solicitado

Dado que Antonio es un profesional

Cuando Antonio consultas las tareas que le han solicitado

Entonces Antonio debería ver las tareas que le han solicitado

9.1.24. Permitir al cliente obtener un informe con las tareas que ha solicitado a determinado profesional

Dado que José es un cliente

Y que Antonio es un profesional

Cuando José consulta las tareas que ha solicitado a Antonio

Entonces José debería ver las tareas que ha solicitado a Antonio

9.1.25. Permitir al profesional obtener un informe con las tareas que le han sido solicitadas por determinado cliente

Dado que Antonio es un profesional

Y que José es un cliente

Cuando Antonio consulta las tareas que le ha solicitado José

Entonces Antonio debería ver las tareas que le ha solicitado José

9.1.26. Permitir al cliente actualizar la valoración dada al trabajo realizado por un profesional

Dado que María es una clienta

Y que Carmen es una profesional

Y que María solicitó una tarea a Carmen

Y que Carmen presupuestó la tarea que le solicitó María

Y que María aceptó el presupuesto

Y que Carmen realizó la tarea

Y que María pagó la tarea

Y que María valoró la tarea

Y que María ha completado la nueva valoración

Cuando María envía la nueva valoración

Entonces a María se le debe informar de que su valoración se ha procesado correctamente

Y la nueva valoración debería sustituir a la antigua valoración en la lista de valoraciones realizadas por María

Y la nueva valoración debería sustituir a la antigua valoración en la lista de valoraciones realizadas a Carmen

9.1.27. Permitir al cliente obtener un informe con el historial de valoraciones de un profesional

Dado que José es un cliente

Y que Antonio es un profesional

Cuando José consulta las valoraciones realizadas a Antonio

Entonces José debería ver las valoraciones realizadas a Antonio

9.1.28. Permitir al cliente obtener un informe con la valoración media de un profesional

Dado que José es un cliente

Y que Antonio es un profesional

Cuando José consulta la valoración media de Antonio

Entonces José debería ver la valoración media de Antonio

9.1.29. Permitir al profesional obtener un informe con su historial de valoraciones recibidas

Dado que Antonio es un profesional

Cuando Antonio consulta las valoraciones que le han realizado

Entonces Antonio debería ver las valoraciones que le han realizado

9.1.30. Permitir al profesional obtener un informe con su valoración media

Dado que Antonio es un profesional

Cuando Antonio consulta su valoración media

Entonces Antonio debería ver su valoración media

9.1.31. Permitir a los profesionales obtener un informe con los productos Premium a la venta

Dado que Carmen es una profesional

Cuando Carmen consulta los productos Premium a la venta

Carmen debería ver los productos Premium a la venta

9.1.32. Permitir a los profesionales la adquisición de alguno de los productos Premium que los hará aparecer más arriba y les otorgará un distintivo en los informes de profesionales cercanos

Este requerimiento funcional se divide en dos historias de usuario.

9.1.32.1. Pago con credenciales de pago válidas

Dado que Carmen es una profesional

Y que Carmen ha indicado que desea adquirir determinado producto Premium

Y que Carmen ha introducido credenciales de pago válidas cuando se le han pedido

Cuando Carmen confirma el pago

Entonces Carmen debería ver un mensaje de agradecimiento

Y el distintivo característico del producto Premium adquirido debería estar visible en el perfil de Carmen

9.1.32.2. Pago con credenciales de pago inválidas

Dado que Carmen es una profesional

Y que Carmen ha indicado que desea adquirir determinado producto Premium

Pero que Carmen ha introducido unas credenciales de pago inválidas cuando se le han pedido

Cuando Carmen confirma el pago

Entonces a Carmen se le debería informar de que las credenciales de pago introducidas son inválidas.

9.2. Pruebas efectivamente realizadas

9.2.1. Pruebas de aceptación realizadas manualmente

Las pruebas de aceptación son pruebas funcionales, y como tales suelen simular la interacción del usuario con el sistema. Comencé a leerme *Learning Android Application Testing* con la esperanza de encontrar para Android una herramienta que hiciera algo parecido a lo que Selenium hace para web, pero que pudiera aprender rápidamente. Encontré la herramienta, pero al tiempo a invertir en aprenderla no me compensaba para este proyecto. De modo que las pruebas de aceptación las he realizado manualmente y con cierta recurrencia.

9.2.2. Pruebas de integración extremo a extremo automatizadas

La estrategia que se ha seguido para probar las APIs que publicaban los diferentes componentes de la aplicación consiste en utilizar pruebas de integración extremo a extremo automatizadas. Esta estrategia tiene la gran ventaja de que es rápida de implementar, y de que implementada de forma exhaustiva es bastante robusta. El único inconveniente es que estas pruebas son lentas de ejecutar.

Estas pruebas están organizadas por controlador, de modo que cada controlador tiene su clase de prueba. Sin duda, el uso de controladores con poca funcionalidad ayuda a evitar que las clases que los prueban crezcan demasiado. Dentro de estas clases de prueba encontramos los métodos de prueba. El contenido de estos métodos suele seguir la misma estructura: preparación de la petición, llamada a la API y comprobación de la respuesta.

Dado que estamos hablando de APIs HTTP REST, los métodos en realidad lo que hacen es aplicar operaciones sobre recursos. De modo que para cada tipo de operación y para cada tipo de resultado debería haber al menos una prueba en una implementación exhaustiva de esta estrategia.

En nuestro caso hemos priorizado comprobar operaciones con resultado exitoso, antes que operaciones con un resultado no exitoso de cualquier tipo, basándonos en las especificaciones de las pruebas de aceptación.

Por ejemplo, en la petición de profesionales cercanos, tenemos una prueba para valores válidos de longitud y latitud, pero no para valores inválidos que simplemente retornarán un error.

Las tecnologías que se han utilizado para implementar esta estrategia son JUnit y Mockito, especialmente por los métodos de sus librerías para Java, y luego Spring que, además de proporcionarnos un ejecutor de pruebas anotadas con JUnit, nos brinda el componente MockMvc que resulta ideal para implementar esta estrategia. MockMvc lo utilizamos para realizar las llamadas a la API y comprobar los resultados.

Por último, decir que no se han realizado implementaciones específicas para las pruebas de las interfaces que se comunican con sistemas externos (PayPal o Firebase Cloud Messaging), y que, evidentemente, quien proporciona las dependencias a nuestras clases de prueba es Spring.

9.2.3. Otras pruebas

Quiero dedicar este apartado a las pruebas de rendimiento y estrés realizadas sobre la consulta de profesionales cercanos.

El punto de partida de dichas pruebas era una tabla con un millón de profesionales. Dichos profesionales tienen una ubicación y una profesión aleatorias pero válidas, y el resto de sus campos, en especial las cadenas de carácteres, tienen la longitud que se espera tendrían en el caso de que los profesionales fueran reales. Todo un ejercicio de simulación muy sencillo.

Lo único que había que hacer era lanzar peticiones de consulta de profesionales a modo de prueba. Pruebas con unas pocas peticiones sirven para comprobar que PostgreSQL + PostGIS operan más deprisa sobre puntos geométricos que sobre puntos geográficos.

Luego fue cuestión de ver qué rendimiento obtenía lanzando peticiones en lotes de 100, esperando a que terminaran las 100 peticiones antes de lanzar las siguientes 100, y ver que en

mi máquina de desarrollo (i5-460M a 2,53 GHz, 4 GB de memoria DDR3, disco duro SATA a 7200 rpm) la velocidad de procesamiento del lote se mantenía más o menos constante, y que se procesaban unas 10 peticiones por segundo. Rara vez fallaba alguna petición de las 100.

Con estos resultados disponibles, la elección de la máquina que servirá nuestro componente geodirectorio de profesionales es mucho más sencilla.

10. Conclusiones

10.1. Conclusiones personales

Durante la conclusión del proyecto he llegado a varias conclusiones, de las cuales en esta sección incluyo y detallo algunas de ellas.

10.1.1. Más simple es mejor

Cuando comencé este proyecto, en mi inexperiencia, no sabía bien qué hacía una solución software mejor que otra. Tenía una noción difusa de lo que era una solución software de calidad: una con una arquitectura que permitiera aislar entre ellas las lógicas de dominio, de aplicación y de infraestructura; un modelo muy cuidado y refactorizado a patrones; implementación de librerías propias orientadas a objetos para permitir una integración más suave con otros sistemas externos.

En cambio, ahora lo tengo claro: la solución software que es mejor, es la más simple. Cuando digo simple, me refiero a simple en todo sentido: simple para los usuarios, simple para el equipo de desarrollo y simple para el equipo de operaciones.

En primer lugar, tiene que ser simple para los usuarios porque la complejidad les incomoda y disgusta. La simplicidad es importante para, por un lado, cautivar su interés inicialmente y, por otro lado, para no perder su interés posteriormente. El éxito a la hora de lograr una implementación simple para el usuario final dependerá principalmente del diseño de la interfaz de usuario y de su usabilidad.

En segundo lugar, tiene que ser simple para el equipo de desarrollo, para que pueda seguir añadiendo funcionalidades a buen ritmo manteniendo la simpleza. Precisamente este es el sentido que tiene el refactorizado: permitir simplificar las cosas y, por tanto, mantener la simplicidad de la solución software. Igual que la simplicidad es importante para el equipo de desarrollo, también lo será para el equipo de mantenimiento: es mucho más fácil corregir defectos en una solución software simple; es precisamente por este motivo que el equipo de mantenimiento debería mantener la simplicidad del sistema cuando lo modifica para, por ejemplo, corregir defectos.

En tercer lugar, tiene que ser simple para el equipo de operaciones. Sin embargo, esta simplicidad a veces depende más de las herramientas, de los procesos y de las buenas prácticas del equipo de operaciones que de la solución software en sí. No obstante, cuando depende de la solución software en sí suele optarse por mantener la simplicidad de la solución para el equipo de desarrolladores, de modo que el equipo de operaciones tiene que adecuarse a la complejidad existente.

10.1.2. La simplicidad es un objetivo al que aspirar

Desconocía cuando comencé este proyecto la verdadera razón de ser de los diferentes estilos arquitectónicos, o de los patrones que se proponían para casi cualquier cosa. Entendía la importancia de mantener los elementos desacoplados, y de mantener los componentes concentrados en su propósito y no asumiendo responsabilidades que no les correspondían. Entendía que un patrón era una solución cuasi óptima para determinado tipo de problemas, y que mantener el código limpio era importante para hacerlo legible y que así resultara más fácil trabajar con él.

Lo que no sabía es que todo eso aspiraba a lo mismo: a la simplicidad, a lograr un código y una solución software sencillos. Exactamente lo mismo a lo que aspira el refactoring: a simplificar el código cada vez que se aplica.

Así por ejemplo, ahora comprendo que un patrón o un estilo arquitectónico no deben aplicarse siempre que sean aplicables, sino solamente cuando su aplicación resulta en una solución más simple. Sin duda, una conclusión importante para mí.

10.1.3. En este proyecto tiene un peso mucho mayor la infraestructura que el dominio

Cuando empecé este proyecto, no sabía la dimensión que iba a adquirir la infraestructura del mismo. La mayor parte de la lógica de infraestructura la componen toda la parte de integración con Firebase Cloud Messaging vía XMPP, y luego también la parte de integración entre componentes, y la parte de integración con PayPal.

Por otro lado, los modelos del dominio son muy pequeños y están muy centrados en un único propósito: permitir la realización de los casos de uso de la aplicación.

10.1.4. La inexperiencia crea falsas expectativas

La inexperiencia en el uso de una tecnología, y la inexperiencia a la hora de planificar, sin duda crean unas muy falsas expectativas. Uno cree que va a ser capaz de aprender a utilizar la tecnología rápidamente, y que cuando comience a usarla va a hacerlo con agilidad y con velocidad.

Nada más lejos de la realidad. En la sección de planificación se pueden encontrar las diferencias entre lo planificado inicialmente y lo que finalmente ha resultado. Y en ella no se incluyen horas extra de trabajo o de formación.

10.2. Revisión de objetivos

Inicialmente dividía los objetivos en tres grupos. Veamos ahora qué objetivos he satisfecho de cada grupo.

En primer lugar, los objetivos de la aplicación eran dos. Por un lado, construir un prototipo no listo para lanzar al público que permitiera al cliente contactar con el profesional, al cliente y al profesional intercambiar mensajes, y al cliente pagar al profesional. Como se verá en la demostración, este objetivo se ha cumplido. Por otro lado, la aplicación debía ser útil, objetivo que sinceramente creo se ha cumplido.

En segundo lugar, los objetivos míos personales eran dos. Por un lado, formarme y aprender lo máximo posible, lo cual he cumplido con creces especialmente en el plano teórico. Por otro lado, el objetivo era la implementación de un sistema complejo y ambicioso, y aunque la implementación no es perfecta, sí es funcional, de modo que considero también cumplido este objetivo.

Los objetivos del proyecto consistían en que se cumplieran los objetivos de los otros dos grupos, y que la memoria documentara correctamente el trabajo realizado. Los objetivos de los otros dos grupos, como se ha visto, han sido todos satisfechos. Sobre si la memoria documenta correctamente el trabajo realizado, esto es algo que deben decidir mi tutor y mi tribunal.

10.3. Ampliaciones y mejoras

Hay ampliaciones y mejoras que ya han sido tratadas extensamente a lo largo de esta memoria, como por ejemplo el sistema invitaciones o el sistema más completo de tareas. También se ha introducido antes el concepto de ludificación o gamification en la sección de diseño de la interfaz de usuario.

En este apartado, se repasarán dichas ampliaciones y mejoras y se verán otras.

10.3.1. Sistema de invitaciones

Un sistema de invitaciones que permita a los usuarios invitar a sus conocidos. Dichas invitaciones contendrían un enlace de registro, que si es usado por alguien para registrarse, entonces el sistema registraría quién fue el invitador del nuevo usuario. Esto daría derecho al invitador a participar en las tasas que aplicara la aplicación a los pagos que efectuase o recibiese el nuevo usuario.

Implementar esto en una aplicación con cliente web sería muy sencillo: sin embargo, cuando el cliente es móvil las cosas se complican. El propio producto Firebase tiene un sistema envío de invitaciones por correo electrónico o SMS, especialmente diseñado para aplicaciones cuyo cliente es móvil.

Dichas invitaciones contienen enlaces especiales que transportan a alguna vista de determinada aplicación móvil. Incluso si la aplicación no está instalada, te lleva a Play Store o al Apple Store para que te la descargues. Una vez abierta la vista a la que apunta el enlace, se puede acceder a información adicional contenida en el enlace, como por ejemplo el

identificador del invitador. En caso de que el usuario se registre, se podría incluir dicha información en la petición de registro.

10.3.2. Un sistema de tareas más completo

Sin duda la implementación de esta ampliación puede que no fuera vista por algunos como una mejora, puesto que quitaría flexibilidad a la aplicación. Para nosotros, sin embargo, sería una fuente de información interesante sobre la que hacer inteligencia empresarial. Habría que valorar pues, si la rigidez añadida resulta en un perjuicio para el usuario, en cuyo caso se descartaría la idea.

Hacerlo más completo significa introducir un sistema rígido de solicitudes, presupuestos, aceptaciones y denegaciones de presupuestos, y poner prerrequisitos a una funcionalidad como los pagos. También habría que modificar la interfaz de usuario móvil para permitir una gestión más eficiente de las tareas.

10.3.3. Ludificación o gamification

Aplicar ludificación o gamification a la aplicación para sustituir el actual sistema de reputación. En vez de ser un sistema basado en los ingresos y las valoraciones, lo que se pretendería con esta idea es mantener las valoraciones, pero sustituir los ingresos por experiencia y niveles, de modo que resultaría en un sistema de reputación más creíble. En este sentido, no solamente los ingresos totales influirían en la experiencia recibida, sino también otros factores como por ejemplo el número de tareas completadas, o la velocidad con que se completan o cobran tareas.

10.3.4. Convertir el micromercado local actual por un micromercado local sin restricciones

En él, uno podría pedir a la cafetería que le subieran un café a la oficina, o bien podría realizar consultas a un médico cercano a cambio de un módico precio, o incluso solicitarle una visita. Las posibilidades son muchas.

11. Bibliografía

Bauer et al. *Java Persistence with Hibernate*. Manning, 2015.

Buschmann et al. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, 1996.

Deinum et al. *Spring Recipes: A Problem-Solution Approach, Third Edition*. Apress, 2014.

Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2003.

Ferguson. *BDD in Action*. Manning, 2014.

Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

Obe and Hsu. *PostGIS in Action, Second Edition*. Manning, 2015.

Millett and Tune. *Patterns, Principles and Practices of Domain-Driven Design*. Wrox, 2015.

Phillips et al. *Android Programming: The Big Nerd Ranch Guide, Second Edition*. Big Nerd Ranch, 2015.

Saint-Andre et al. *XMPP: The Definitive Guide*. O'Reilly, 2009.

Schildt. *Java The Complete Reference, Ninth Edition*. Oracle Press, 2014.

Schmidt et al. *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. Wiley, 2000.

Shigeoka. *Instant Messaging in Java: The Jabber Protocols*. Manning, 2001.

Vernon. *Implementing Domain-Driven Design*. Addison-Wesley, 2013.

Videla and Williams. *RabbitMQ in Action: Distributed Messaging for Everyone*. Manning, 2012.

[Blanco]

http://ocw.unican.es/ensenanzas-tecnicas/gestion-de-proyectos-software/ejercicios/Lista%20de%20Riesgos%20en%20proyectos%20SW.pdf