DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 1

Wi-Find Bar

Ingeniería del Software I+II

Grupo 7

Integrante	LU	Correo electrónico
Ciruelos Rodríguez, Gonzalo	063/14	gonzalo.ciruelos@gmail.com
Costa, Manuel José Joaquín	035/14	manucos94@gmail.com
Gatti, Mathias Nicolás	477/14	mathigatti@gmail.com
Maddonni, Axel Ezequiel	200/14	axel.maddonni@gmail.com
Thibeault, Gabriel	114/13	gabriel.eric.thibeault@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

Este trabajo se basó en el diseño e implementación de una aplicación que permita a sus usuarios buscar bares cercanos a ellos con Wi-Fi y enchufes.

Cómo modelo de desarrollo de software utilizamos Scrum, que es un proceso iterativo incremental (considerado Agile). En el trabajo se presenta el resultado de los dos primeros sprints, de una duración de alrededor de dos semanas cada uno. Nuestro rol fue tanto de desarrolladores como de Product Owners, por ejemplo porque tomamos las decisiones claves de diseño por nuestra cuenta, y creamos las user stories y los requerimientos nosotros mismos.

En cuanto a la implementación, utilizamos el lenguaje de programación Python. Python es un lenguaje interpretado y dinámicamente tipado. Además, soporta varios paradigmas de programación, entre ellos el orientado a objetos. Esto se va a reflejar en una (a veces significativa) diferencia entre el código y el diseño que veremos más adelante. Esto se debe a que los lenguajes dinámicamente tipados son más flexibles, entonces no es necesario implementar interfaces o clases 100 % abstractas.

Como framework para crear la aplicación utilizamos Flask [1], que es muy poderoso y muy pequeño a la vez, lo cual nos permitió que no interfiera a la hora de diseñar, y presentar el diseño en diagramas.

[1] http://flask.pocoo.org/

2. Planificación y seguimiento

En esta sección analizamos primero generalidades sobre la planificación y luego entramos a ver en detalle el Sprint Backlog de cada iteración. A cada uno lo sigue un breve apartado sobre el seguimiento de la respectiva iteración, mostrando el burndown chart correspondiente. Finalmente realizamos la retrospectiva.

2.1. Comentarios preliminares sobre las User Stories

Es un requerimiento que la comunidad sea la encargada de subir los bares. Sin embargo, es de esperar[2] que los usuarios eventualmente agreguen bares repetidos, inexistentes, con información errónea o abusen del sistema de diversas formas (e.g. spam). Consecuentemente, decidimos agregar un nivel de indirección, es decir que un usuario en vez de subir un nuevo bar, realiza una sugerencia a un Moderador (que puede aceptarla o rechazarla); éstos pueden ser miembros del equipo o usuarios distinguidos. Al verificar las sugerencias mediante un Moderador buscamos incrementar la calidad de los resultados.

El sistema de sugerencias a Moderadores se implementará no sólo para agregar nuevos bares, sino para modificar la información de bares ya existentes o para reconocer a los dueños de un bar. Incorporamos el concepto de dueño de un bar, mediante el cual un usuario puede ser reconocido como propietario de un local, y puede actualizar la información del establecimiento sin requerir verificación de un Mod.

Ya que introducimos el concepto de Moderadores y de dueños de bares, algunos usuarios deben poder reconocerse. Por ende, determinamos que los usuarios podrán tener una cuenta a la cual se loguean para utilizar ciertas funcionalidades de la aplicación. Esto nos presentó un dilema: cómo tratar a los usuarios que no tienen cuentas? Propusimos tres soluciones básicas: forzar a los usuarios a crear una cuenta y loguearse a ésta para utilizar la aplicación; permitir que los usuarios utilicen la aplicación "deslogueados", y que se logueen a sus cuentas para acceder a las funcionalidades que lo requieran; loguear automáticamente a los usuarios no registrados a una cuenta default (e.g. "Guest" o "Anónimo"), y que se logueen para acceder a las funcionalidades correspondientes.

La primera opción nos pareció poco recomendable: las aplicaciones que requieren crear cuentas inmediatamente sin razón son molestas, y preferimos que los usuarios puedan usar la plataforma sin tener que loguearse. Las otras dos opciones son esencialmente idénticas para el usuario; sin embargo, la tercera nos permite simplificar el código ya que podemos asumir que los usuarios siempre están logueados a una cuenta. Debido a esto, fue la que elegimos.

Para las features de los bares (e.g. calidad del wi-fi, cantidad de enchufes, precio) decidimos emplear una calificación de 5 estrellas, con una granularidad de media estrella[3]. Tener sólo 5 calificaciones posibles nos pareció muy restrictivo, por lo que adoptamos una granularidad de media estrella. Discutimos sobre tomar 0 o media estrella como calificación mínima; decidimos adoptar media por claridad, pues una valoración de 0 estrellas puede ser confundida con que nadie haya calificado aún la feature.

En lo que concierne a la presentación de los bares más cercanos, decidimos ordenarlos siempre por distancia. Adicionalmente, el usuario puede aplicar filtros por feature, eliminando

[2] En base a nuestra experiencia personal navegando por internet.

[3] Las calificaciones posibles son $\{\frac{1}{2}, 1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 4, \frac{9}{2}, 5\}$.

resultados que no cumplen ciertos criterios. Éstos pueden ser, $\leqslant, \geqslant$, o por rango cerrado (e.g. si la calificación $\in [2,4]$)[4].

La User Story a la que le asignamos 1 Story Point es a "Borrar Comentarios", ya que consiste sólo en agregar un botón a la interfaz de comentarios y removerlo de la estructura que lo contiene, ambas tareas muy sencillas. Para decidir el resto de los Story Points de los User Stories utilizamos la técnica de Planning Poker.

2.2. Roles de usuario

Consideramos que la siguiente elección de roles nos da una granularidad suficiente para nuestro problema:

- usuario que busca un bar: un usuario cuyo objetivo es encontrar un bar.
- usuario que critica un bar: un usuario que quiere criticar un bar, lo cual puede hacerlo a través de comentarios y calificaciones.
- usuario con cuenta y usuario sin cuenta: son respectivamente los roles para usuarios que están o no registrados, y que realizan acciones relacionadas a este sistema de registro.
- moderador: usuario con privilegios especiales que le permiten agregar, editar y eliminar todos los bares.
- dueño de bar: usuario con privilegios especiales para editar o eliminar el o los bares que posee.
- usuario que usa Facebook y usuario que usa Twitter: son los roles asociados a la interacción con redes sociales.

2.3. Primer Sprint: Sprint Backlog

Para el primer Sprint elegimos las User Stories esenciales para implementar la funcionalidad básica de buscar un bar cercano: la búsqueda en sí, la vista de bares y sus features, agregar bares (sólo como Moderador, ya que es más simple y con esto alcanza en una primera instancia), el log-in[5] y la promoción a Moderador. Estas dos últimas son necesarias ya que un bar sólo puede ser agregado por un Moderador.

Buscar bares

como	usuario que busca un bar
quiero	buscar bares cerca de mí
para	poder conocer qué bares cercanos tienen wifi y enchufes.

Criterio de aceptación: Al enviar la posición actual del usuario al sistema, este debe devolver la lista de bares a menos de 400m.

[4] Para algunas features, como precio, se podrán utilizar todos estos filtros; para otras, sólo algunos: no tiene sentido buscar sólo bares con mala calidad de wi-fi.

[5] Si bien podremos loguearnos a cuentas ya existentes, aún no podremos crear nuevas. Hardcodearemos algunas, y nos loguearemos a éstas.

Business Value: 10 *Story Points:* 8 (*RoI:* 1.25)

Tasks:

1. Crear página de búsqueda. (2)
2. Crear la función de búsqueda, que toma como parámetro la posición del usuario. (3)
3. Devolver una lista de jsons para cada bar con su nombre, su distancia, su valoración y un link a su página dentro de la aplicación. (1)

   Total de horas hombre: 6

Agregar bares

como	moderador
quiero	poder agregar un nuevo bar
para	que pueda ser sugerido por la aplicación a los usuarios

Criterio de aceptación: El moderador debe poder cargar los datos y crear un bar. El bar debe aparecer en las búsquedas posteriores, reflejando su existencia en el sistema.

Business Value: 8 *Story Points:* 5 (*RoI:* 1.6)

Tasks:

1. Añadir botón en el menú de acciones del moderador para agregar a la base de datos un nuevo bar. (1)
2. Hacer que al presionar un botón se cargue un formulario para rellenar con los datos del bar. (3)
3. Hacer una función que dado un formulario cree una entrada en la base de datos. (2)

   Total de horas hombre: 6

Borrar bares

como	moderador o dueño del bar
quiero	poder borrar un bar existente
para	que no pueda ser sugerido por la aplicación a los usuarios

Criterio de aceptación: Un bar del catálogo debe poder ser seleccionado por el moderador o el dueño del bar. Si un usuario busca el bar eliminado, éste no deberá aparecer. Si un usuario está puntuando o comentando cuando se elimina, recibirá un error. Si dos moderadores intentan borrar simultáneamente el mismo bar, el último en hacerlo recibirá un error.

Business Value: 5 *Story Points:* 2 (*RoI:* 2.5)

Tasks:

1. Agregar botón en la página del bar para que pueda ser borrado por un moderador o su dueño. (2)
2. Hacer una función que lo elimine de la base de datos. (1)

   Total de horas hombre: 3

Vista del bar

como	usuario que busca o critica un bar
quiero	poder acceder a la página de un bar en la aplicación
para	ver sus características con mayor detalle

Criterio de aceptación: La vista del bar debe contener todos los datos actualizados: nombre, foto, ubicación, features y comentarios.

Business Value: 10 *Story Points:* 3 (*RoI:* 3.33)

Tasks:

1. Agregar un template para la página correspondiente a la vista de un bar. (3)
2. Agregar una función para traer los datos actualizados del bar en un json desde la base de datos. (1)

   Total de horas hombre: 4

Log-in

como	usuario con cuenta
quiero	loguearme
para	acceder a funcionalidades disponibles sólo a usuarios con cuenta.

Criterio de aceptación: Al ingresar un usuario y contraseña válidas, el sistema debe iniciar una sesión con ese usuario.

Business Value: 8 *Story Points:* 5 (*RoI:* 1.6)

Tasks:

1. Crear un sistema de log-in. (3)
2. Crear una página de log-in para que los usuarios puedan ingresar su usuario y contraseña. (2)

   Total de horas hombre: 5

Promover a moderador

como	moderador
quiero	dar privilegios de moderación a un usuario con cuenta
para	permitir que un usuario confiable pueda ser moderador.

Criterio de aceptación: Si el usuario seleccionado existe, se le deberán otorgar privilegios de moderación y tendrá disponibles todas las herramientas de los moderadores a partir de ese momento.

Business Value: 5 *Story Points:* 3 (*RoI:* 1.66)

Tasks:

1. Hacer función que le de privilegios de moderación a una cuenta. (1)

   Total de horas hombre: 1

2.4. Primer Sprint: Seguimiento

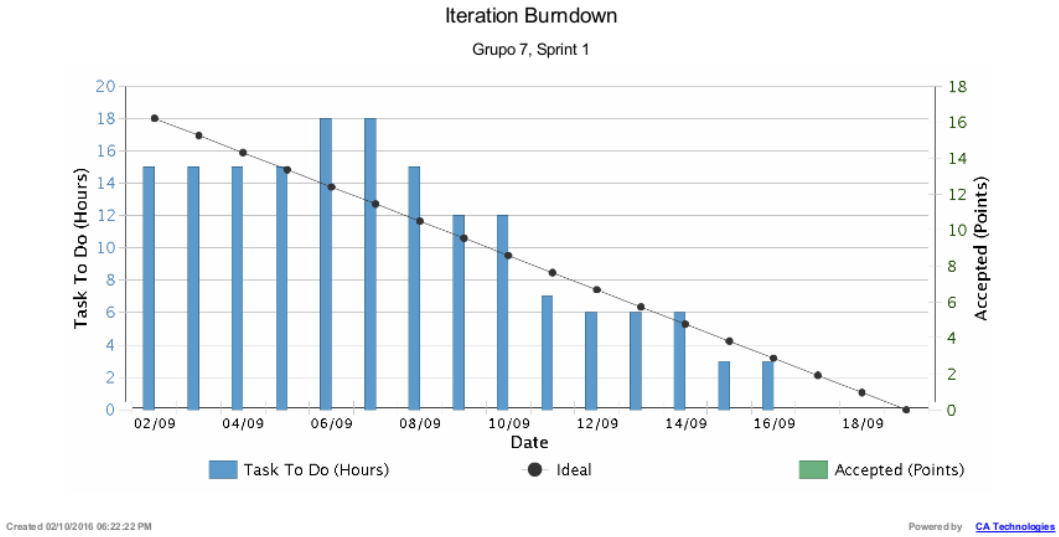

Figura 1

La subida en la cantidad de horas restantes que puede observarse el 06/09 se explica porque se nos había pasado por alto asignarle la estimación de horas a la User Story "Vista de bar" al momento de crearla. El 06/09 fue cuando corregimos dicho error.

En general quedamos muy conformes con el flujo de trabajo durante este primer sprint. Como puede verse en el gráfico logramos mantenernos al día, cumpliendo con todos los objetivos propuestos.

2.5. Segundo Sprint: Sprint Backlog

En el segundo Sprint buscamos expandir significativamente el proceso de búsqueda y vista de bares, incorporando filtros por feature y distancia, permitiendo editar información

de bares (nuevamente, sólo como Moderador), calificarlos, comentar y obtener el trayecto óptimo hasta el bar. También agregaremos la opción de crear cuentas, ya que la alternativa de hardcodearlas es engorrosa.

Editar información de bares

como	moderador o dueño del bar
quiero	poder editar la información de un bar del sistema
para	actualizar datos incompletos, incorrectos o desactualizados.

Criterio de aceptación: Dado un bar, el dueño o un moderador debe tener un botón para editarlo dentro de la página del bar. Luego de editarlo, si alguien accede la página del bar, éste aparecerá con sus datos actualizados. Si dos moderadores editan un bar al mismo tiempo, la edición definitiva será la del que lo envíe segundo.

Business Value: 6 *Story Points:* 3 (*RoI:* 2)

Tasks:

1. Agregar botón en el menú de acciones que tiene el moderador/dueño del bar sobre cada bar para que pueda editar la información. (1)
2. Hacer que al presionar el botón se cargue un formulario con la información actual del bar, la cual podrá ser editada y actualizada en la base de datos al presionar un botón de confirmación. (4)

   Total de horas hombre: 5

Votación

como	usuario que critica un bar
quiero	poder calificar a un bar con una cantidad de estrellas del $\frac{1}{2}$ (peor) al 5 (mejor) en todas sus features
para	dar una valoración rápida y concreta que ayude al resto de la comunidad en sus futuras búsquedas.

Criterio de aceptación: Cuando un bar es calificado por un usuario, el nuevo voto deberá verse reflejado cuando la página del bar sea cargada posteriormente.

Business Value: 9 *Story Points:* 2 (*RoI:* 4.5)

Tasks:

1. Modificar la representación de los bares para que pueda almacenar el promedio de votos de cada categoria y la cantidad de votos de cada categoria. (2)
2. Hacer que las funciones que leen la información de un bar lean también las calificaciones. (2)

   Total de horas hombre: 4

Comentar

como	usuario que critica un bar
quiero	escribir comentarios sobre los bares que visito
para	compartir detalles que considere importantes para ayudar al resto de la comunidad en futuras búsquedas.

Criterio de aceptación: Al escribir y aceptar un nuevo comentario, éste debe aparecer en la vista del bar.

Business Value: 7 *Story Points:* 2 (*RoI:* 4.5)

Tasks:

1. Modificar la representación de los bares para que pueda almacenar una lista de comentarios. (1)
2. Modificar la página de vista de los bares para que puedan mostrarse los últimos 10 comentarios. (2)
3. Agregar un botón para que puedan verse todos los comentarios de un bar. (2)

Total de horas hombre: 5

¿Cómo llegar?

como	usuario que busca un bar
quiero	conocer la ruta más rápida para llegar al bar deseado desde mi posición actual
para	minimizar mi pérdida de tiempo y esfuerzo.

Criterio de aceptación: En la vista de un bar deberá haber un botón que permita ver el camino al bar desde mi posición. Éste deberá ser efectivamente el camino óptimo.

Business Value: 9 *Story Points:* 3 (*RoI:* 3)

Tasks:

1. Agregar un botón en la vista de un bar que permita acceder al mapa de camino más rápido. (1)
2. Interactuar con la API de Google Maps para obtener el mapa. (3)

Total de horas hombre: 4

Filtrar por distancia

como	usuario que busca un bar
quiero	filtrar los resultados de una búsqueda de bares por máxima distancia
para	buscar y distinguir bares según la distancia a la que se encuentran

Criterio de aceptación: Al modificar la preferencia de distancia máxima de búsqueda, los bares resultantes deberán encontrarse a menor o igual distancia que la seleccionada.

Business Value: 8 *Story Points:* 2 (*RoI:* 4)

Tasks:

1. Agregar opción de filtro por distancia a la lista de resultados. (1)
2. Agregar función de filtrado por distancia y filtrar los resultados usándola. (3)

Total de horas hombre: 4

Filtrar búsquedas por feature

como	usuario que busca un bar
quiero	filtrar los resultados de una búsqueda según el puntaje de las distintas features de los bares
para	poder descartar de los resultados los bares que tengan una puntuación indeseable en ciertas categorías

Criterio de aceptación: Al seleccionar un filtro, los resultados de la búsqueda ejecutada que aparecen deben pasar las condiciones impuestas por los filtros, y deben ser todos los que las cumplen.

Business Value: 8 *Story Points:* 3 (*RoI:* 2.66)

Tasks:

1. Hacer función que chequee si un bar pasa los criterios especificados. (2)
2. Filtrar los resultados que se le muestran al usuario utilizando dicha función. (2)

Total de horas hombre: 4

Creación de cuenta

como	usuario sin cuenta
quiero	crear una cuenta
para	poder loguearme y acceder a funcionalidades sólo disponibles a usuarios logueados.

Criterio de aceptación: Al ingresar un usuario y contraseña y confirmar la contraseña, si el usuario no existe en el sistema, se creará un nuevo usuario, que podrá loguearse de ahí en adelante.

Business Value: 9 *Story Points:* 3 (*RoI:* 3)

Tasks:

1. Hacer una página de creación de usuarios. (1)
2. Hacer función que cree usuario. (1)

Total de horas hombre: 2

2.6. Segundo Sprint: Seguimiento

2.7. Retrospectiva

Si bien en estas dos primeras iteraciones del proyecto pudimos desarrollar el núcleo de las funcionalidades relacionadas con la búsqueda de bares, la aplicación aún dista mucho de estar completamente terminada. Quedan un gran número de cosas por hacer:

- algunas que son claves pues hacen a los requerimientos iniciales de la aplicación, como por ejemplo que los usuarios puedan sugerir nuevos bares;
- otras que apuntan más a un refinamiento de lo que ya hay, como hacer más cómoda la búsqueda de bares evitando que el usuario tenga que volver a buscar un bar cada vez que retorna de ver un perfil, o que los usuarios con cuenta tengan un historial de comentarios y calificaciones;
- y también extender la aplicación con otras funcionalidades (no centrales inicialmente) que pueden ser claves en la mecánica del negocio, como interactuar con redes sociales.

A continuación presentamos las User Stories que actualmente nos quedan en el backlog.

Volver a página de resultados

como	usuario de la aplicación
quiero	poder volver a los resultados obtenidos en una búsqueda desde la vista de un bar seleccionado a partir de la misma
para	poder acceder a las vistas de otros bares obtenidos en la búsqueda sin tener que volver a realizarla

Business Value: 5 *Story Points:* 3 (*RoI:* 1.66)

Sugerir un nuevo bar

como	usuario con cuenta
quiero	sugerir que se agregue un nuevo bar
para	que pueda ser aprobado por un mod y de esta forma sea agregado al catálogo

Business Value: 7 *Story Points:* 5 (*RoI:* 1.4)

Sugerir dueño de un bar

como	usuario con cuenta
quiero	proponerme como dueño de un bar que ya se encuentra en el sistema
para	ser reconocido como dueño del bar y poder actualizar su información libremente.

Business Value: 5 *Story Points:* 5 (*RoI:* 1)

Aprobar la creación de un bar

como	moderador
quiero	poder aprobar o rechazar una sugerencia para agregar un nuevo bar
para	para que la sugerencia sea procesada y removida de la cola.

Business Value: 5 *Story Points:* 5 (*RoI:* 1)

Aprobar dueño

como	moderador
quiero	poder aprobar o rechazar la sugerencia de un usuario que se indica como dueño de un bar
para	para que la sugerencia sea procesada y removida de la cola.

Business Value: 5 *Story Points:* 5 (*RoI:* 1)

Remover comentarios

como	moderador
quiero	remover comentarios
para	hacer cumplir las condiciones de uso de la aplicación.

Business Value: 3 *Story Points:* 1 (*RoI:* 3)

Interactuar con Facebook

como	usuario que usa Facebook
quiero	compartir el perfil del bar en mi muro de Facebook
para	mis contactos de Facebook conozcan el bar y sepan mi opinión al respecto.

Interactuar con Twitter

como	usuario que usa Twitter
quiero	twittear el perfil del bar
para	mis contactos de Twitter conozcan el bar y sepan mi opinión al respecto.

2.8. Modificaciones respecto de la primer entrega

Se realizaron las siguientes modificaciones menores a partir de las correcciones recibidas tras la primer entrega de las user stories:

- Tratamos de ser más realistas en cuanto a lo que falta por hacer, no dar "la sensación de tener toda la aplicación terminada en dos sprints".

- Pasamos de tener un único rol genérico para los usuarios a varios más específicos.
- Agregamos al backlog User Stories relacionadas con la interacción con redes sociales.

3. Diseño Orientado a Objetos

3.1. Diseño del sistema

Primero veamos como se comporta el sistema en general. Para eso, utilizaremos un diagrama de clases.

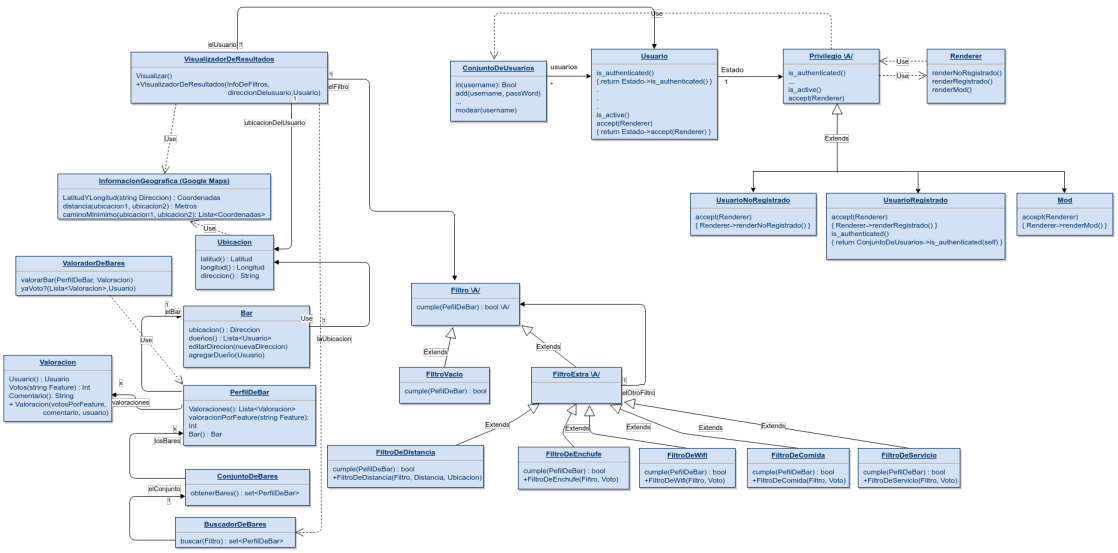

Figura 2: Diagrama de clases del sistema completo.

Analicemos y justifiquemos el diagrama parte por parte.

Ubicación Va a ser la clase que nos va a abstraer de las diferentes posibles representaciones para la ubicación de un bar, en particular el String que representa su dirección, y su latitud y longitud que va a ser útil para calcular distancias, y en general para interactuar con la API de Google Maps, que vamos a usar fuertemente.

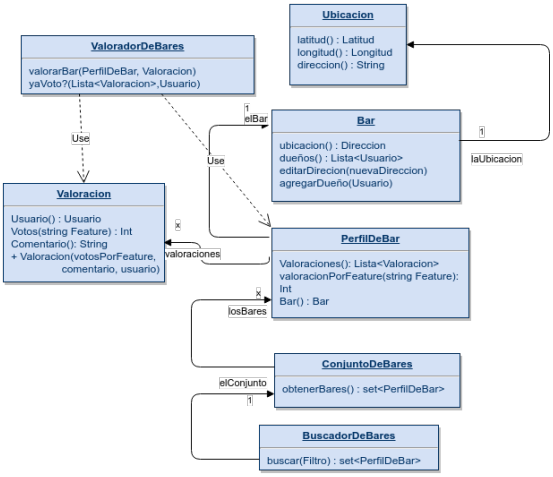

Figura 3: Diagrama de clases del subsistema de bares.

Bar Va a ser la clase que va representar a los bares de la realidad. Los bares tienen una ubicación y una lista de dueños.

PerfilDeBar Es la clase que nos permite representar un bar desde el punto de vista de la aplicación. Para nuestra aplicación un bar es más que simplemente una dirección y una lista de dueños: un bar tiene votos y comentarios. La clase PerfilDeBar debe pensarse como simplemente un wrapper para la clase Bar, que contiene información pertinente a la aplicación.

ConjuntoDeBares Representa a todos los bares que existen en la aplicación.Existe porque nos pareció mejor tener una clase separada que abstraiga esta idea, que tener simplemente un conjunto de bares hardcodeado adentro de BuscadorDeBares.

BuscadorDeBares Es la clase que nos va a permitir buscar bares. Tiene como colaborador interno un ConjuntoDeBares que es el conjunto sobre el cual va a buscar. Recibe un único mensaje que es buscar, que toma como parámetro un filtro.

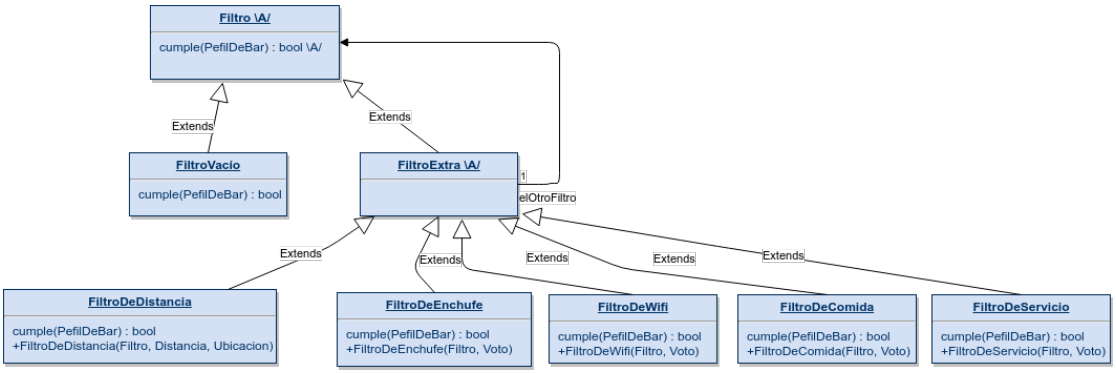

Figura 4: Diagrama de clases del subsistema de filtro.

Filtro Para la clase filtro vamos a utilizar el patrón Decorator. Filtro es una clase abstracta, que tiene un mensaje llamado cumple, que recibe un PerfilDeBar y chequea que cumpla su

condición. En caso de que la cumpla, va a devolver True, en caso que no, False. Al final del trabajo hablamos más en detalle sobre porqué tomamos las desiciones que tomamos.

FiltroVacio Va a ser el filtro trivial que nos va a permitir finalizar la composición de filtros, como indica el patron Decorator. Su implementación de cumple va a ser `return True;`.

FiltroExtra Va a ser la clase abstracta que nos va a permitir componer filtros. Nótese que tiene como colaborador interno otro Filtro, entonces las clases que hereden de filtro extra, en su implementacion de cumple van a tener que chequear que su condición se cumpla, y que la condición del colaborador interno se cumpla.

FiltroDeX Van a ser las clases concretas que nos van a permitir filtrar por las diferentes características de un PerfilDeBar. Un comentario importante: la signatura de cumple en nuestra implementación de FiltroDeDistancia difiere un poco de la presentada en el diagrama, pero tiene que ver con temas implementativos de la API de Google Maps que no pudimos evitar; aunque debe tenerse en cuenta que la idea de alto nivel es la misma.

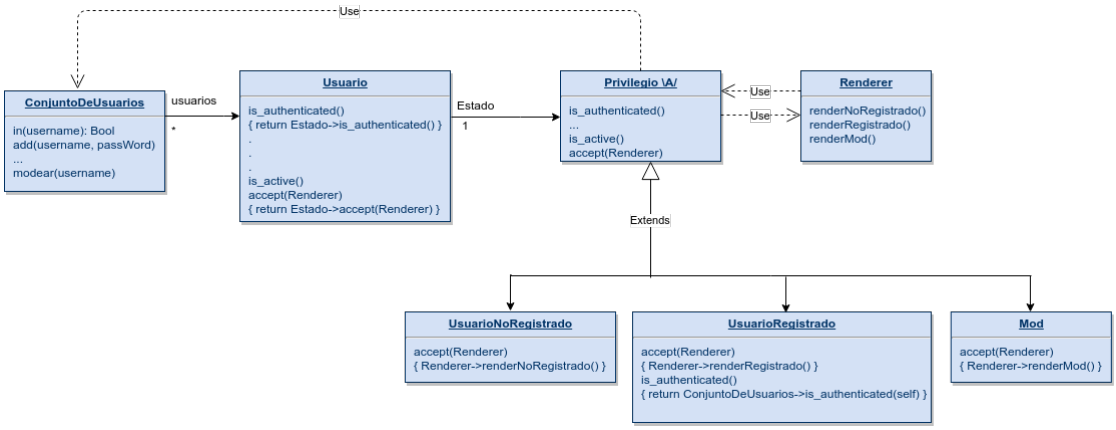

Figura 5: Diagrama de clases del subsistema de filtro.

ConjuntoDeUsuarios Como su nombre lo indica, representa el conjunto de usuarios del sistema. Adicionalmente, esta clase es la encargada de validar las credenciales provistas para loguearlos y autenticarlos.

Implementativamente, se accede mediante un *Singleton*.

Usuario Ésta es la clase que representa a un usuario. Los usuarios pueden tener distintos privilegios[6], y éstos pueden cambiar en *runtime*; para contemplar esto, empleamos el patrón de diseño *State*[7].

Cabe destacar que la clase usuario presenta ciertos mensajes[8] que deberían ser responsabilidad de ConjuntoDeUsuarios; éstos deben ser implementados por la clase Usuario para poder emplear el módulo *Flask-Login*. Nuestros métodos forwardean dichos mensajes a ConjuntoDeUsuarios, ya que son su responsabilidad.

[6] Actualmente, un usuario puede ser No-Registrado, Registrado o Mod.
[7] Empleando la terminología de "*Design Patterns*" de Gamma et al, Usuario es el *Context* del patrón.
[8] Por ejemplo, `is_authenticated()`.

Privilegio Como mencionamos, la clase Usuario emplea el patrón *State*; Privilegio es la clase abstracta de la cual heredan los privilegios concretos. En términos de "*Design Patterns*, Privilegio es el *State* del patrón.

Mod, etc... Éstos son los privilegios concretos que puede adquirir un Usuario; son los *Concrete States* del patrón. Como tales, los métodos de Usuario simplemente forwardean los mensajes a estas clases, que se ocupan de realizar la implementación propriamente dicha.

Renderer El rendering de HTML al cargar una página[9] puede depender de los privilegios del usuario actual, pero no es responsabilidad del Usuario.

Para resolver esto, empleamos un patrón *Visitor*: Usuario presenta un mensaje *accept*, cuyo método forwardea el mensaje a Privilegio, y cada privilegio concreto llama al mensaje correspondiente del Renderer[10].

Cabe destacar que empleamos un único *visitor* concreto, en vez de una jerarquía de *visitors* concretos que heredan de uno abstracto, pues requerimos uno solo. Sin embargo, dado el *duck typing* de *Python*, es trivial refactorear nuestra implementación a una con dicha jerarquía, en caso de requerirlo en el futuro.

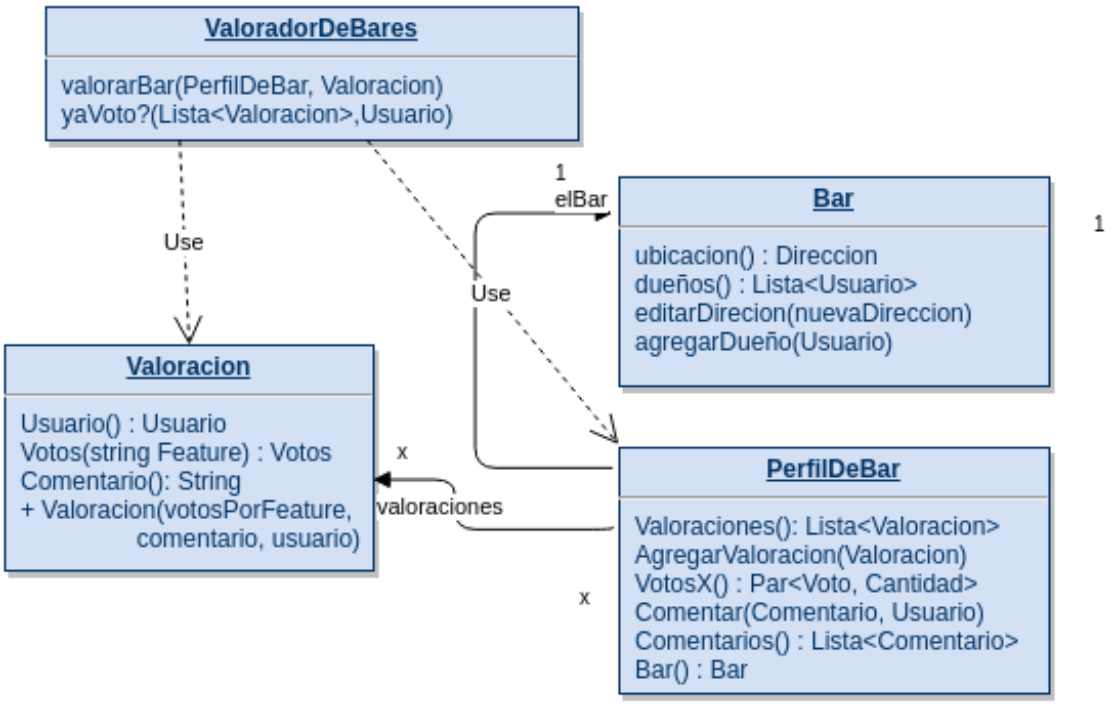

Figura 6: Diagrama de clases del subsistema de Valoración.

Voto Es un multiplo de 0.5 entre 0.5 y 5 inclusive.

[9] Podría haber más eventos con requerimientos similares; actualmente el rendering de HTML es el único que se nos presentó, por lo que nos concentraremos en éste. Sin embargo, lo siguiente es aplicable a cualquiera de éstos.

[10] Que corresponde al *visitor* del patrón, en términos de "*Design Patterns*".

Valoración Es un objeto que contiene un conjunto de votos sobre distintos features (Wifi, Enchufes, etcétera), un comentario y el usuario que realizo los mismos. Como decisión de diseño escogimos impedir que un usuario pueda votar sobre un único feature, de esta manera una Valoracion es siempre completa ya que contiene un puntaje asignado para todos los atributos actuales y un comentario.

ValoradorDeBares Es el objeto encargado de manejar el sistema de valoración. Se creó con el objetivo de aumentar la cohesión en nuestro diseño y quitarle responsabilidades relacionadas al chequeo de las normas de valoración a los bares (Verificar que el usuario no haya votado previamente, entre otras que podrían agregarse, como chequear que este logueado, que no sea dueño del bar, etcétera) lo cual en nuestra opinión no modelaba la realidad.

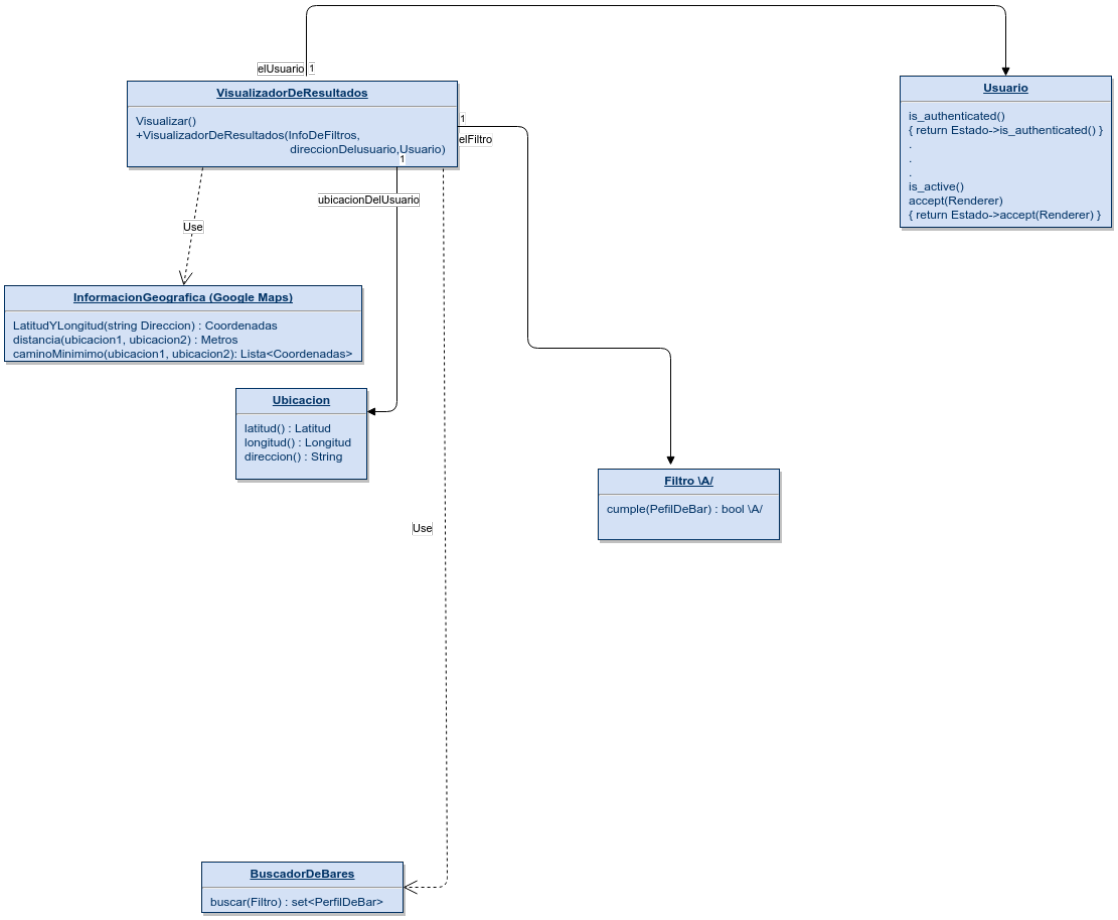

Figura 7: Diagrama de clases del subsistema de visualización de resultados.

VisualizadorDeResultados Esta clase tiene el propósito de generar la visualización de resultados de bares. El mensaje de creación de instancia se ocupa de recibir un formulario de búsqueda directo de la request http POST, y generar los filtros adecuados. Además, tiene un método llamado Visualizar que simplemente se ocupa de buscar y generar el HTML de la página de re-

sultados. En la siguiente sección veremos con diagramas más detallados como interactúa todo.

3.2. Diagramas de Objetos y Secuencia

3.2.1. Calificar bares

En esta sección mostraremos los diagramas correspondientes a la valoración de un bar. Aquí se puede observar como una valoración realizada por Juan Perez para el bar Niceto es enviada al Valorador para que chequee si Juan Perez esta habilitado a Valorar (Actualmente chequea unicamente si el usuario no voto previamente en el mismo bar) y en caso afirmativo agrega la valoración al $PerfilDelBar$.

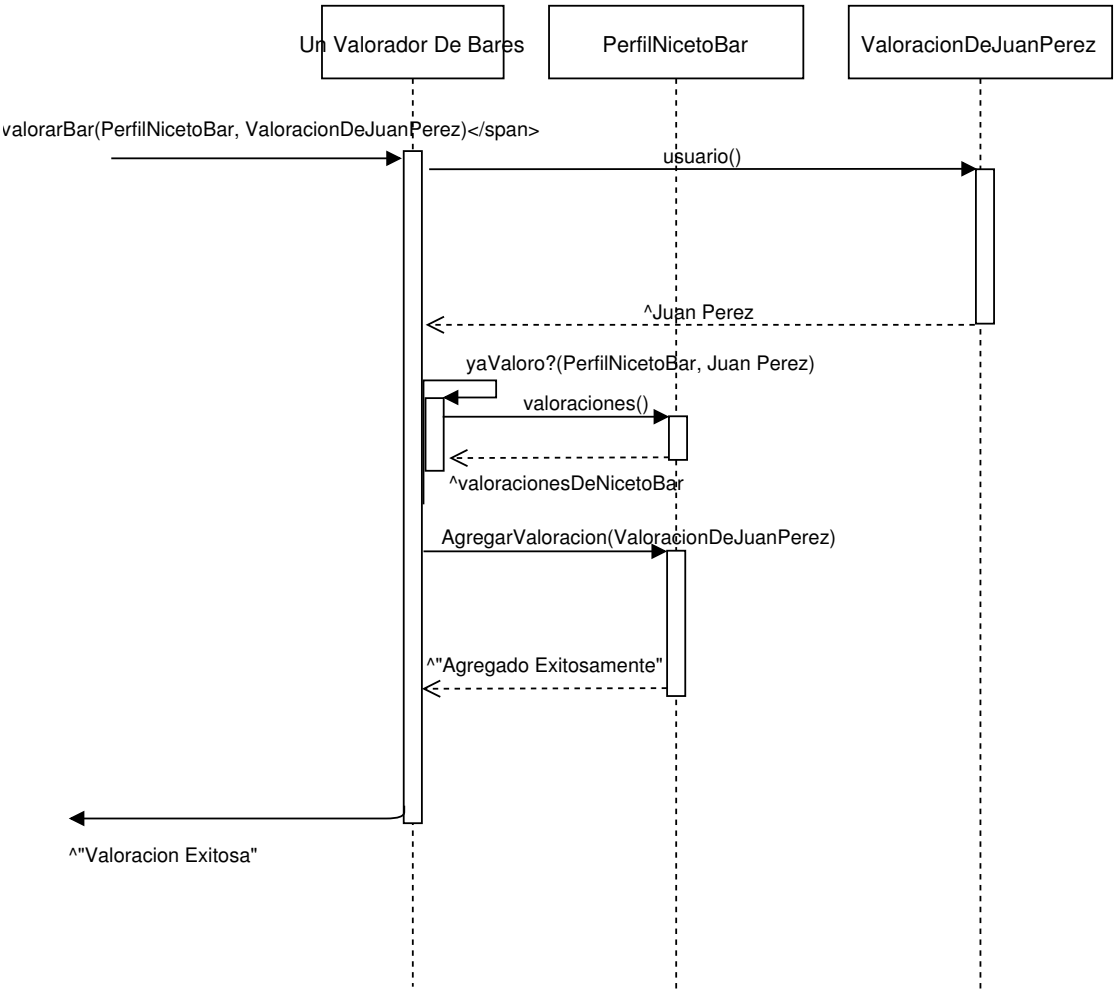

Figura 8: Diagrama de secuencia.

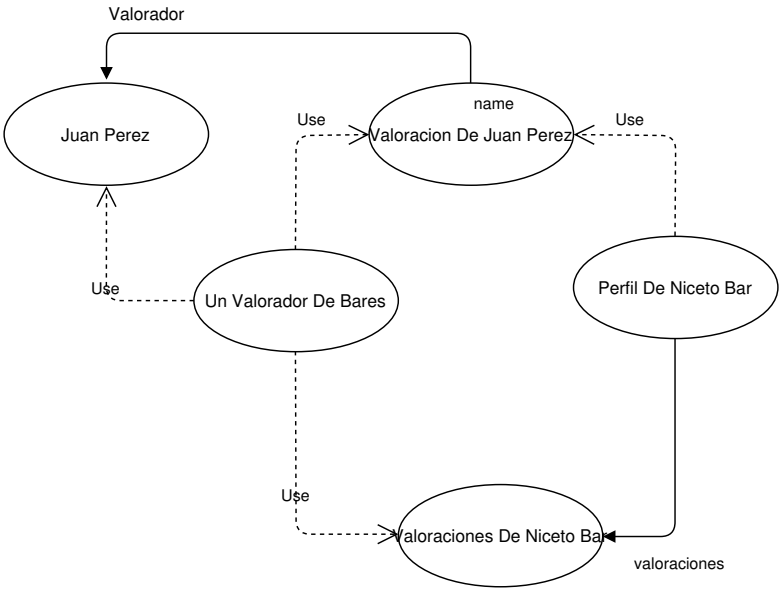

Figura 9: Diagrama de objetos.

3.2.2. Filtro de bares filtrando por una única característica

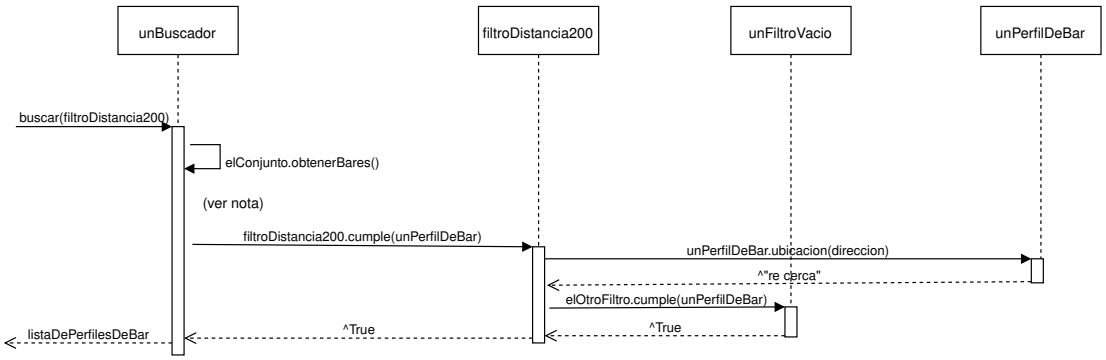

Figura 10: Diagrama de secuencia. Nota: corremos, `filter(bares, filtroDist200.cumple)`, por simplicidad ponemos sólo una llamada a cumple. Además, donde dice "votos", debería decir "valoracionPorcentualPorFeature", pero lo escribimos así para hacerlo más corto y que el diagrama no quede mal.

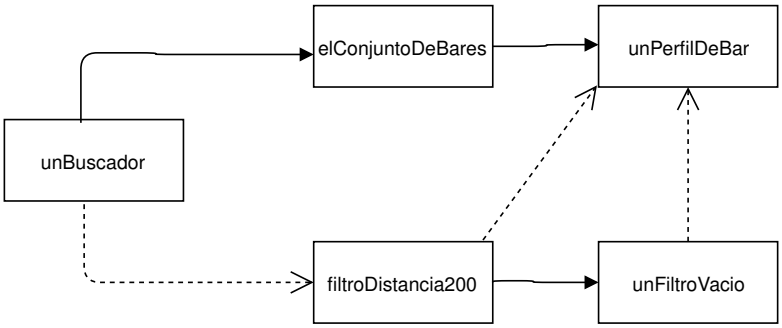

Figura 11: Diagrama de objetos.

Cómo se ve claramente, el filtro es un típico ejemplo de uso de Decorator. Como se ve, unBuscador recibe un conjunto de perfiles de bares. Como aclaramos en la nota, lo que hacemos es simplemente llamar a `filter`, pero ejemplificamos lo que sucede con un perfil específico. En este caso, la distancia del bar es menor que 200, entonces devuelve True.

3.2.3. Filtro de bares filtrando por tres característica

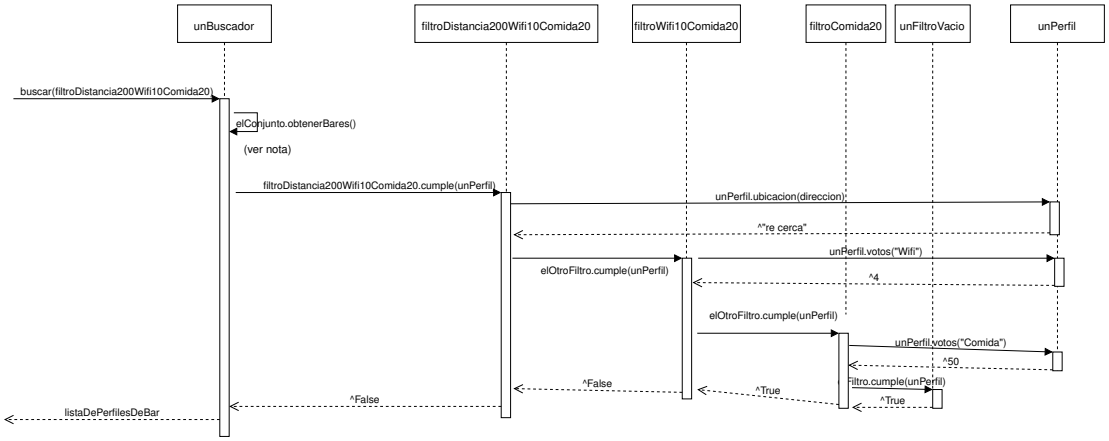

Figura 12: Nota: `corremos filter(bares, filtroDist200Wifi10Comida20.cumple)`, por simplicidad ponemos sólo una llamada a cumple. Además, donde dice "votos", debería decir "valoracionPorcentualPorFeature", pero lo escribimos así para hacerlo más corto y que el diagrama no quede mal.

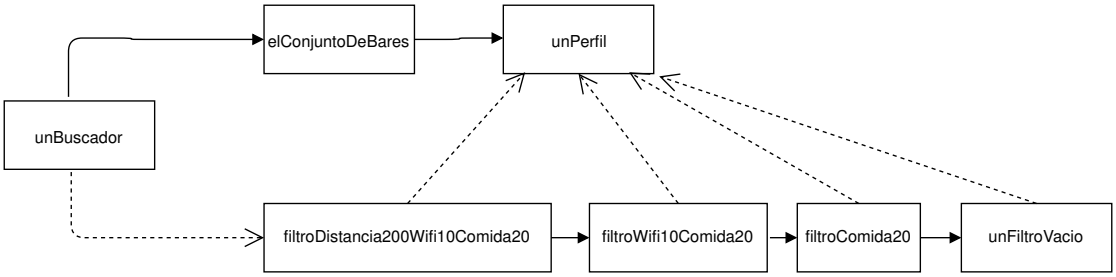

Figura 13: Diagrama de objetos.

Igual que arriba, se ve que es un típico ejemplo de uso de Decorator.

Como aclaramos en la nota, lo que hacemos es simplemente llamar a `filter`, pero ejemplificamos lo que sucede con un perfil específico. En este caso, la distancia al bar es menor que 200, pero la valoración del Wifi es 4, que es menor que 10, con lo cual esa llamada va a devolver False, valor que se va a propagar hacia atrás.

En cuanto al diagrama de objetos es igual que antes. El buscador tiene adentro al conjunto de bares que a su vez tiene adentro a unPerfil. Sin embargo, los filtros solo conocen a los perfiles que le llegan como parámetro.

3.2.4. Visualización de resultados

Ahora pasaremos a ver la parte de visualización de resultados. Esta parte fue modelada con una clase llamada VisualizadorDeResultados, que permite encapsular la parte de generación de resultados. En los diagramas no mostramos la parte "implementativa" que es la que se ocupa de llamar a Flask para generar el HTML porque creemos que no suma.

Sin embargo, mostramos la parte más vital, que es la de creación de filtro y de búsqueda de resultados. Veamos como es.

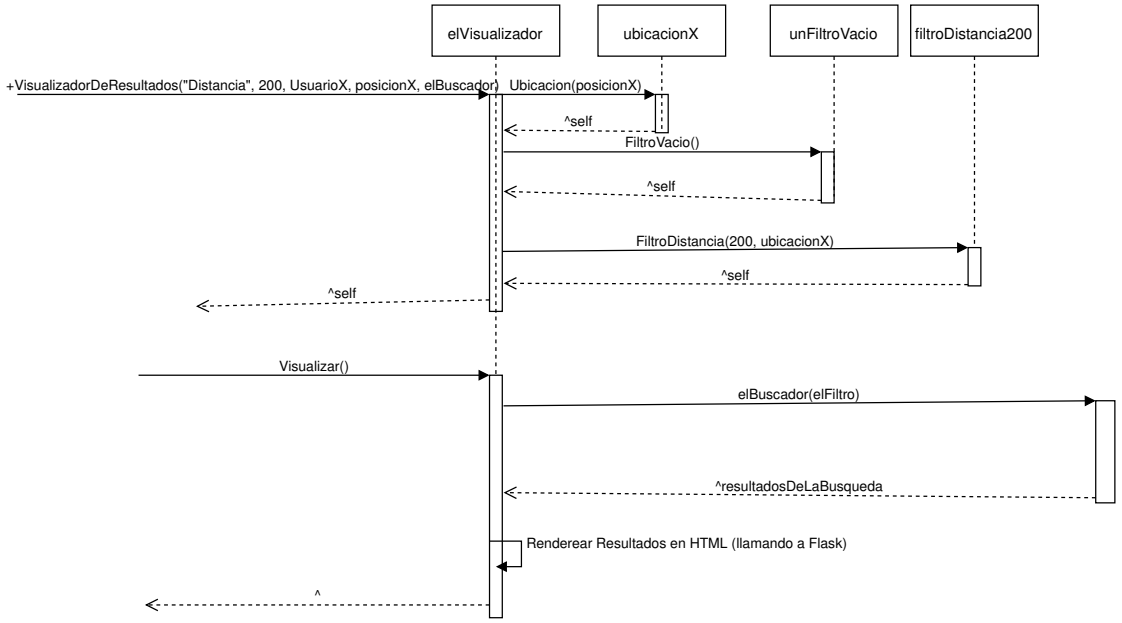

Figura 14: Diagrama de secuencia.

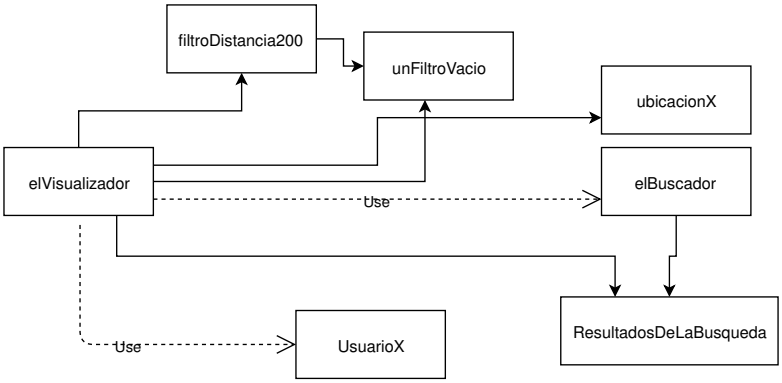

Figura 15: Diagrama de objetos.

Como se ve, hay muchos objetos que interactúan. Esto no quiere decir que el acoplamiento sea alto, debido a que los objetos que interactúan tiene sentido que interactuen para mostrar los resultados.

3.2.5. Autenticar Usuario

A continuación presentamos los diagramas de objetos y secuencia de una llamada (exitosa) a `ElConjuntoDeUsuarios.autenticar(UnUsername, UnaPassword)` durante el proceso de *log-in*.

Si bien el proceso de *log-in* es más extenso que la llamada a `autenticar`, lo omitido está

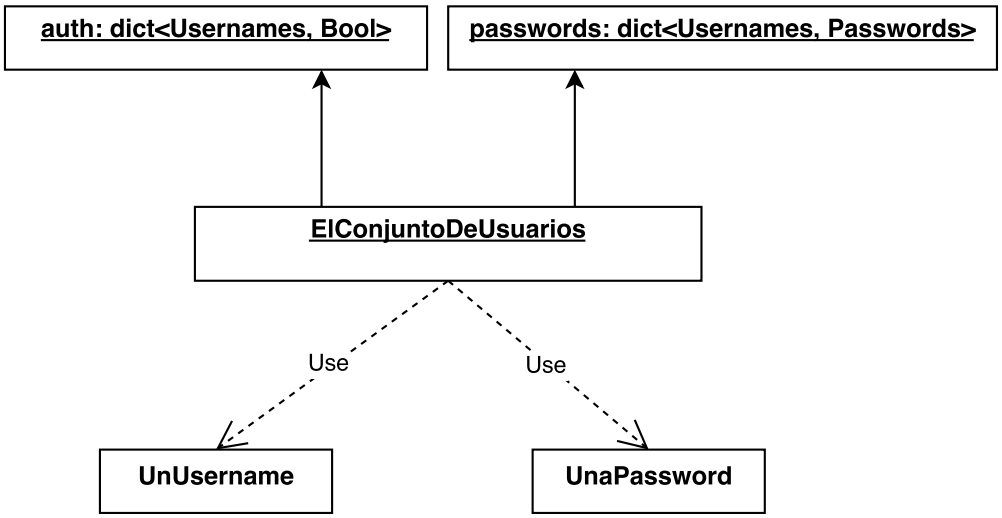

Figura 16: Diagrama de objetos de una autenticación (exitosa).

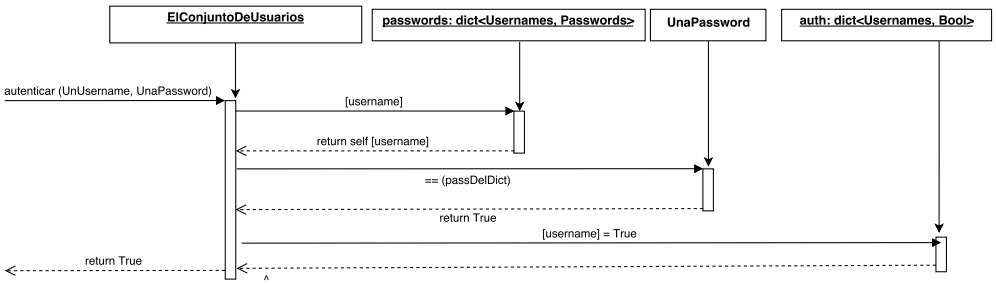

Figura 17: Diagrama de secuencia de una autenticación (exitosa).

relacionado a *Flask*, rendering de html, llenado de *forms* y demás; tales detalles son ortogonales al diseño planteado, por lo que sólo presentamos dicha llamada.

Se muestra un caso exitoso; si falla la búsqueda en el diccionario (al no existir el usuario) o la comparación entre *passwords*, se levanta una excepción que la función de *log-in* captura y reacciona correspondientemente.

3.3. Más en detalle…

3.3.1. Incorporación de bares

3.3.2. Incorporación de nuevas características de bares

Como nosotros tomamos la decisión de diseño de que las features no sean "positivas o negativas", si no porcentuales, la feature que agregaríamos valoraría la calidad de los espectáculos, y en caso de que no haya, valdrá 0.

Agregar esto no sería nada difícil, sólo habría que agregar un campo mas a los features

de la clase Valoración (además de cambiar los templates HTML para que muestren la nueva información).

En caso que se quisiera filtrar además por la feature de los espectáculos en vivo, se deberá agregar una nueva clase que herede de FiltrosExtra y filtre por esta categoría.

Como se ve, las modificiaciones involucran sólo agregar código (y modificar Valoración, que es correcto que deba modificarse), con lo que el diseño es resiliente frente a este eje de cambio.

3.3.3. El diseño y comportamiento de los distintos filtros de búsqueda de bares

Recordemos cómo era el diagrama de clases de los filtros.

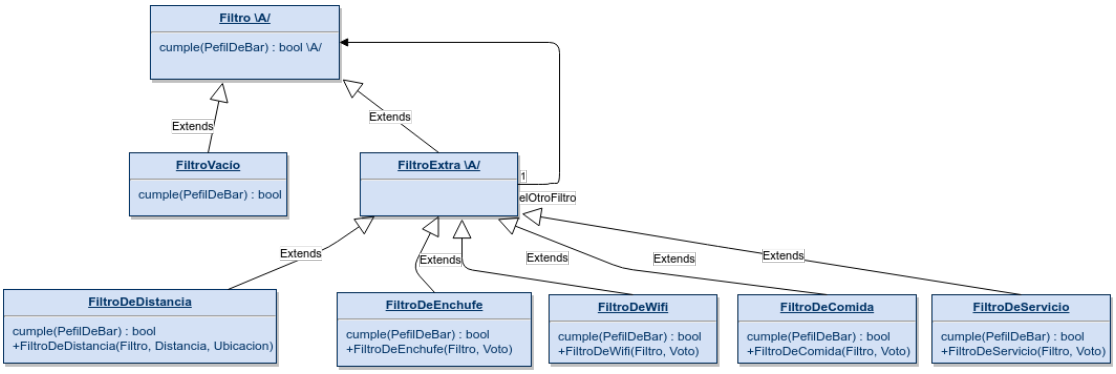

Figura 18: Diagrama de clases.

Como dijimos anteriormente, los filtros los diseñamos usando el patrón Decorator.Elegimos este patrón porque parecía el más natural para resolver el problema y efectivamente lo era.

Recordemos que el "intent" del patrón Decorator es "Asignar responsabilidades a un objeto dinamicamente y proveer una alternativa a la subclasificación para extender", que es exactamente lo que queremos hacer aquí, queremos, dinámicamente, poder crear un filtro de distintas features dinámicamente.

En cuanto al funcionamiento, fue explicado intensamente en la sección anterior, con dos diagramas de objetos y dos diagramas de secuencia que dejan en claro como van a interactuar los objetos.

Luego de finalizar el diseño y la implementación, notamos que quizás todos las subclases de FiltroExtra (excepto FiltroDeDistancia), podrían contraerse a una única clase.

Sin embargo, esto no formaba parte del sprint, con lo cual la refactorización de este código quedará para un sprint futuro. Además, los Product Owner indicaron que el diseño de filtro era el esperado, con lo cual quizás ni siquiera haya que cambiarlo.

Los filtros podrían ser contraídos a un FiltroDeFeature, que toma como parámetro el string de un feature, y le pide ese feature a las valoraciones del perfilDeBar pasado como parámetro, y las compara. Esto permitiría ahorrar repetición de código, quizas perdiendo expresividad, por eso no nos parece un cambio del todo positivo, pero nos pareció importante decir que tuvimos en cuenta que esto era una alternativa y decidimos no hacerla.