Ingeniería de Software II

Sistema "El precio justo"

Grupo 6

1º Cuatrimestre 2013

LU	Nombre	email
667/06	Daniel Foguelman	dj.foguelman@gmail.com
767/03	Hernán Modrow	hmodrow@gmail.com
511/00	Leonardo Tilli	leotilli@gmail.com
836/02	Paula Verghelet	pverghelet@gmail.com

1. Introducción

Se nos solicita el desarrollo de una herramienta que, para poder brindarle a la comunidad información sobre los mejores lugares y precios para comprar los productos que necesite dentro de la ciudad autónoma de Buenos Aires, permita mediante el revelamiento y selección de tweets lograr tal objetivo.

Se espera que dicha información sea correcta, precisa, independiente, y libre de segundas intenciones. Se sugiere que la forma de lograr este objetivo, es que la propia comunidad sea la que informe sobre los mejores precios.

Es decir, si cada uno va twitteando dónde compró un determinado producto a un muy buen precio, sería posible aprovechar esa información para ayudar a otras personas a cuidar su economía.

La aplicación se enfocará en productos de primera necesidad, los cuales incluyen: zanahorias, zapallitos, papas, tomates, leche, yogurt, manzanas, bananas, crema para rulos, asado, vacío, pan, harina, aceite, azúcar y yerba. Aunque es deseable prever su utilización para informar sobre todo tipo de productos.

A partir de información volcada previamente en Twitter se espera obtener el precio y lugar de venta de estos productos, permitiendo que la aplicación pueda sugerir al usuario dónde es más conveniente comprar el/los productos de su necesidad.

Se plantean inicialmente dos estrategias: caminando lo menos posible y gastando lo menos posible. El conjunto de estrategias puede ser ampliado posteriormente.
El usuario podrá consultar por un rango de precios para los productos y obtener la información adecuada a este tipo de solicitud.

Se espera que dicha información pueda mostrarse en un mapa de manera de facilitar la interacción con el usuario.

2. Parte I

En esta primera parte se realiza un análisis del problema que permite la especificación de requerimientos y la identificación de user stories. Esto nos permite realizar una estimación de las mismas y seleccionar aquellas pertinentes a un primer sprint.

En las siguientes secciones se documenta el Product Backlog y el Sprint Backlog obtenido, así como una breve descripción de los ejes de discusión y las decisiones tomadas sobre el alcance del trabajo a realizar.

2.1. Desarrollo

Al momento de generar las users stories nos pasaron dos cosas, algunas las generamos con mucho nivel de detalle y otras incorporaban cosas más allaá de lo requerido por el TP en sí.
Las primeras, al ver que nos quedaban de una sola tarea, las fuimos agrupando en users stories más generales y pudimos generar tareas adicionales transversales a las mismas.
Sobre las otras, las replanteamos de forma que queden como users stories de investigación y análisis a fin de poder reflejar aquellas cosas que fuimos discutiendo durante esta primera etapa que nos parecieron cosas que uno analizaría en un proyecto real, pero que lamentablemente requieren un mayor análisis, al menos a nivel diseño, del que nos es posible realizar durante el primer sprint. Las reflejamos en el backlog, tal como lo haríamos si continuarámos en el proyecto a fin de saber sobre que cosas deberíamos trabajar en los próximos sprints. También son temas no queríamos incluir en el diseño a entregar para evitar que este se vuelva de tal magnitud que nos fuera imposible de realizar durante el sprint.

2.2. Product Backlog

Se han dejado aquí todas las users stories que entendemos están por fuera del alcance de los objetivos del sprint inicial, es decir poder generar una demo. La idea de dicha demo, entendemos, es poder mostrar que es posible procesar la información de los tweets y utilizarla como resultado de las búsqueda.
Las users stories US63, US53, US59, US54 y US70 son las users stories de investigación y análisis mencionadas en la sección anterior.

ID	Descripción	Est	Value
US93	Como usuario quiero buscar por producto para poder elegir el lugar/precio que mejor se ajuste a mi necesidad.	8	7
US74	Investigar tecnologías de desarrollo	3	2
US75	Investigar en la api de twitter para poder estimar las tareas de código	5	3
US72	Generar los diseños necesarios del sistema	8	2
US115	Como sistema quiero identificar y extraer la información útil en un tweet para poder utilizarla en las búsquedas.	5	7
US114	Generar datos para la demo	1	1
US93	Como usuario quiero buscar por producto para poder elegir el lugar/precio que mejor se ajuste a mi necesidad.	8	7
US42	Armar UI para soportar funcionalidades básicas	5	4
US116	Como sistema quiero poder identificar productos con errores de escritura para no descartar tweets de productos habilitados.	5.00	4
US44	Como usuario quiero buscar por zona para realizar las compras cercanas a un lugar de mi elección	8.00	5
US45	Como usuario quiero buscar un producto en un rango de precios para poder seleccionar productos de acuerdo a lo que estoy dispuesto a pagar por el mismo	3.00	5
US46	Como usuario quiero buscar un conjunto de productos para minimizar las consultas puntuales, y poder elegir el lugar/precio que mejor se ajuste a mi necesidad.	3.00	6
US47	Como usuario quiero minimizar el costo total de los productos buscados para realizar un ahorro en la compra total	5.00	5
US48	Como usuario quiero minimizar el recorrido para minimizar el tiempo utilizado en la compra	21.00	5
US51	Como usuario quiero ver todos los productos, con sus precios, asociados a un punto de venta para saber que otros productos se venden a buen precio en dicho punto de venta	5.00	4
US52	Como usuario quiero ver los resultados de una búsqueda en un mapa para ubicarme fácilmente en relación al lugar en el que me encuentro	13.00	4
US55	Como administrador quiero administrar productos soportados para poder agregar quitar o editar los productos indexados	5.00	3
US63	Investigar beneficios e inconvenientes de incorporar distintos formatos de dirección	21.00	2
US53	Investigar beneficios e inconvenientes de que la aplicación tenga usuarios	21.00	3
US59	Investigar beneficios e inconvenientes de identificar subproductos como productos diferenciados	21.00	1
US54	Investigar beneficios e inconvenientes de que el producto mantenga estadísticas de uso	21.00	4
US70	Investigar beneficios e inconvenientes de que el producto mantenga un sistema de veracidad/ranking de la información	21.00	3
US119	Investigar beneficios e inconvenientes de incorporar distintas formas de referencia a productos.	21.00	2
US123	Como usuario quiero buscar puntos de venta de un producto por cercanía a un lugar para poder elegir los más cercanos a dicho lugar.	21.00	5

2.3. Tareas

El siguiente es el listado de tareas que registramos durante el sprint, varias de ellas fueron generadas durante el mismo (lamentablemente no las tenemos distinguidas).

User Story	**Tarea**	**Descripcion**	**Tiempo Estimado**	**Dueño**
US72	TA120	Generar DOO	8	Daniel F
US72	TA121	Generar Diagrama de Clases	8	Modrow
US72	TA122	Generar Diagramas de Secuencia.	8	Modrow
US72	TA123	Otra documentación	6	Modrow
US74	TA102	Investigar opciones para estrategias de testing.	2	Modrow
US74	TA118	Investigar opciones de lenguaje y plataforma	8	Modrow
US74	TA119	Evaluar métodos de acceso a la información de configuración.	1	Modrow
US74	TA188	Crear ambiente de desarrollo local	8	Leonardo T
US75	TA115	Investigación y Prototipado de Autenticación con Twitter	4	Leonardo T
US75	TA116	Investigación y Prototipado de Búsqueda en Twitter.	8	Leonardo T
US75	TA117	Modelo de persistencia de la información de un tweet.	6	Leonardo T
US94	TA133	Diseño y maquetado (anteriormete parte de la US42)	12	Paula V
US93	TA72	Diseñar interfaz de búsqueda para un producto	2	Daniel F
US93	TA73	Diseñar interfaz de resultados del producto buscado	2	Daniel F
US93	TA74	Obtener Productos Habilitados.	2	Paula V
US93	TA75	Búsqueda de un producto	6	Leonardo T
US93	TA132	Implementar interfaz de consulta de Tweets.	8	Leonardo T
US114	TA131	Generar datos de tweets	4	Leonardo T
US115	TA124	Reconocer y extraer producto de un tweet	4	Modrow
US115	TA125	Reconocer y extraer precio de un tweet	2	Modrow
US115	TA126	Reconocer y extraer unidad de un tweet	2	Modrow
US115	TA127	Reconocer y extraer calle y altura de un tweet	4	Modrow
US115	TA128	Reconocer y extraer georeferencia de un tweet	2	Modrow
US115	TA129	Reconocer hashtag en un tweet	1	Leonardo T
US115	TA130	Aplicar validaciones y descartar tweet inválido	4	Modrow

NOTA: La tarea TA133 entendemos debería formar parte de las tareas de la US93, dado que se elimina la US42 por poder adaptarla al esquema de User

Story correctamente.

2.4. Sprint Backlog

ID	Descripción	Est	Value
US93	Como usuario quiero buscar por producto para poder elegir el lugar/precio que mejor se ajuste a mi necesidad.	8	7

Criterios de Aceptación

Caso 1
1. El sistema obtiene una oferta de tomate.
2. El usuario ingresa a la web de "El Precio Justo".
3. El usuario ingresa tomate en el formulario de busqueda.
4. El usuario cliquea en buscar y la búsqueda es enviada al sistema.
5. El usuario puede ver la oferta obtenida por el sistema en el paso 1.

Caso 2
1. El sistema obtiene una oferta de leche a $5.
2. El usuario ingresa a la web de "El Precio Justo".
3. El usuario ingresa leche y con precio menor a $6 en el formulario de busqueda.
5. El usuario cliquea en buscar y la búsqueda es enviada al sistema.
6. El usuario puede ver la oferta obtenida por el sistema en el paso 1.

ID	Descripción	Est	Value
US115	Como product owner quiero que el sistema obtenga inforamción de ofertas de Tweeter para los usuarios puedan buscar dicha información en "El Precio Justo".	5	7

Criterios de Aceptación

1. Se general el tweet "Tomate $6 kilo Libertad 600 #PrecioJusto".
2. El sistema lee los tweets con #PrecioJusto de Twitter y captura el tweet generado en el paso 1.
3. El sistema extrae "tomate" como producto del tweet del paso 1.
4. El sistema extrae "6" como precio del tweet del paso 1.
4. El sistema extrae "kilo" como unidad del tweet del paso 1.
5. El sistema extrae "Libertad 600" como dirección del tweet del paso 1.
6. Genera un oferta con los datos obtenidos de 3,4,5 y 6 para que pueda ser buscada.

ID	Descripción	Est	Value
US74	Investigar tecnologías de desarrollo	3	2

Criterios de Aceptación

1. Decidir cuáles son los lenguajes y tecnologías que se van a utilizar para la implementación de las user stories del sprint
2. Generar una estrategia de testing adecuada para disminuir la cantidad de bugs generados y aumentar la cantidad de bugs detectados
3. Tener un modelo para persistir información de configuración
4. Identificar cuáles son los beneficios e inconvenientes de la utilización de dichas tecnologías

ID	Descripción	Est	Value
US75	Investigar en la api de twitter para poder estimar las tareas de código	5	3

Criterios de Aceptación
1. Obtener un análisis de la correcta utilización de la API teniendo en cuenta autenticación y búsqueda 2. Obtener un análisis de la factibilidad de un modelo para poder persistir información de tweets

ID	Descripción	Est	Value
US72	Generar los diseños necesarios del sistema	8	2

Criterios de Aceptación
1. Generar todos los diagramas necesarios para explicar correctamente el funcionamiento del sistema.

ID	Descripción	Est	Value
US114	Como desarollador quiero generar datos para la demo para tener información que mostrar durante la demo	1	1

Criterios de Aceptación
1. Los datos deberán ser tweets para poder probar la aplicación 2. Los datos deberán cubrir casos de filtrado por producto, dirección, precio, unidad y hashtag 3. Los datos deben permitir la búsqueda sobre tres productos y las unidades asociadas a los mismos para que se puedan realizar búsquedas

NOTA: Se eliminó la US42 por no poder adaptarla al esquema de User Story, la tarea que estaba vínculada se asoció a la US93. Si bien esto no se reflejó en el sprint, entendemos que es como debería haber armado.

3. Parte II

En esta segunda parte describiremos el resultado del sprint, incluyendo las secciones seguimiento, product increment, retrospectiva y de diseño orientado a objetos.

La misma es una revisión de la entrega anterior del documento de acuerdo a las correcciones y anotaciones realizadas.

3.1. Seguimiento

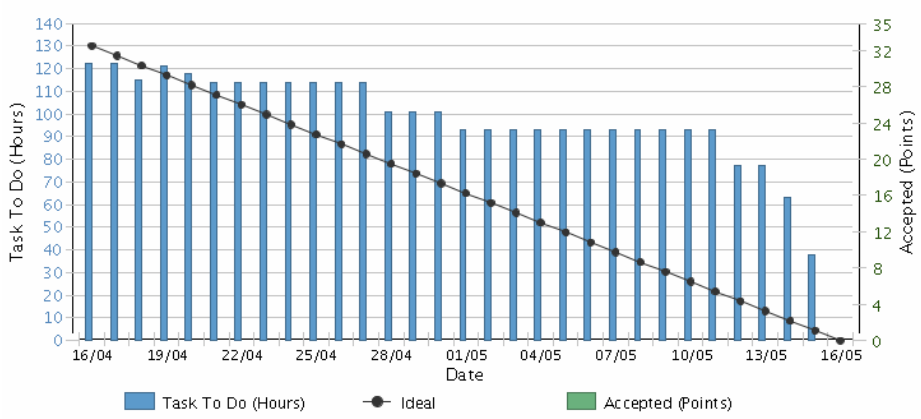
3.1.1. Beneficio del framework Scrum

La utilización de Scrum para el manejo del proyecto nos brindó muchos beneficios en la primer etapa de anális siendo una herramienta de mucha ayuda al descomponer los requerimientos en user stories y luego en tareas.

Sin embargo, al no poder realizar actualización de horas retroactivas, algo bastante común de realizar en otras herramientas de seguimiento, el burnout chart no es fiel reflejo de nuestro avance. Esto nos sucedió ya que al mantener un constante intercambio de mails nos dió la visibilidad del avance suficiente. Posiblemente, creemos, esto no hubiera sucedido de tener que reportar periódicamente el avance a alguien más, como nos sucede en el ámbito laboral.

Adicionalmente, la utilización de la asignación de tareas por intermedio de RallyDev no nos resultó beneficiosa. Principalmente porque las asignaciones iniciales las fuimos cambiando durante el sprint dependiendo de la necesidad de avanzar sobre ciertos punto y de los tiempos disponibles de cada uno, con lo cual andar reflejando constantemente los cambios hubiera resultado en un mayor trabajo.

3.1.2. Burndown chart



3.2. Product Increment

3.2.1. Arquitectura

Durante el análisis generamos diversas User Stories de investigación, estas nos permitieron hacer un análisis de la arquitectura antes de comenzar a implementarla. Con pruebas de concepto que nos permitieron entender la factibilidad de dicha implementación.

Los resultados de dicho análisis se encuentra al final de esta sección, en los siguientes parrafos haremos un análisis de la tecnología elegida.

3.2.2. Lenguaje

La decisión de utilizar un lenguaje de tipado dinámico fue tácita, quisimos evitar las restricciones semánticas que nos impone un lenguaje de tipado estático. También buscamos un lenguaje de Objetos Puro, dejando Python afuera de esta decisión. Finalmente optamos por Ruby, dinámicamente tipado, de objetos puro, con closures.

No es casualidad que la elección del lenguaje nos afectara también en la elección de las tecnologías de soporte de aplicación:

- Servidor SQL, para el soporte de datos de backend
- RESTFul server, para la interfaz de aplicación
- Engine de templating para las vistas

3.2.3. Servidor SQL

Utilizamos SQLite 3 por su portabilidad y facilidad de instalación.

3.2.4. Server

Para brindar los servicios al usuario, utilizamos HTML con una api HTTPRest. Nuestro servidor es Sinatra. De fácil utilización solo es necesario conectar los servicios HTTP a la capa del controlador. ¿Por qué decimos que es REST? Básicamente no depende del estado, es orientado a cliente-servidor y está orientado en capas.

Esto nos permite desacoplar la aplicación de la interfaz de usuario, en la versión actual entregamos una interfaz modesta en HAML, un lenguaje de templating cuya transormación a HTML es nativa en Ruby + Sinatra.

3.2.5. Modelo

Para el modelo de aplicación aplicamos una versión reducida de MVC.

Por un lado tenemos el servidor de aplicación (sinatra como comentabamos en la sección anterior) y por otro lado tenemos un controller cuyas responsabilidades se ven acotadas a la validación del input de usuario y la delegación del procesamiento a ciertas acciones definidas en la capa de servicios.

Los servicios que actualmente soporta el controlador son los de búsqueda.

3.2.6. Extensibilidad

Nuestro diseño como veremos (y justificaremos) en las siguientes secciones, es extensible en:

- Tipos de filtros: nuevos tipos de búsquedas podrán agregarse sin modificar ni la validación de los parámetros ni la interfaz de usuario. Esto lo conseguimos mediante una sencilla técnica de metaprogramación.
- Tipos de consumo de datos de Twitter: actualmente soportamos la api on-line y la api off-line.
- Estratégias de parsing de twitts: en esta versión acotada utilizamos una extracción de datos posicional por medio de regular expressions, esto podrá ser modificado para utilizar un interprete de lenguaje natural que extraiga estos datos cuya ambigüedad es tan variada.
- Interfaces de usuario: actualmente brindamos una interfaz HTML pero podría interactuar con una UI distinta, braile, de escritorio, distribuida, etc.

3.3. Retropesctiva

Principalmente disfrutamos de aprender Ruby, siempre es bueno aprender un lenguaje nuevo.

Creemos que las dificultados no estuvieron en el aprendizaje de nuevas herramientas, si no más bien en tratar de mantener el ritmo de trabajo.

Mención aparte merece el tema del trabajo en equipo. Acá nos encontramos con algunas dificultades, muchas por desconocimiento de cómo trabaja el otro. Ya que salvo Daniel y Hernan el resto nunca había trabajado en un TP juntos. Algunos miembros se adaptaron al dinámica de grupo que se estableció y otros no.

Esto creemos que afectó en la entrega dado que en la recta final algunos miembros del grupo se sobrecargaron de tareas.

También hicimos mucho foco en las herramientas y la implementación y no tanto en el diseño formal. Nos dejamos llevar por el entusiasmo por lo nuevo, si bien estamos convencidos de que el diseño que pensamos es bueno.

Este foco por la implementación y el ritmo tardío de trabajo hizo que nos comunicáramos poco con el product owner para validar nuestro avance. Tal vez acá nos podría haber llamado un poco más la atención, como entendemos que hubiera sucedido en un escenario real (si bien sabemos que ya estamos grandecitos).

En siguientes sprints, creemos que mantener una mayor comunicación con el product owner nos ayudará a mantener cierto ritmo de trabajo constante, para así no tener que realizar un sobresfuerzo para mantener la velocidad ante eventualidades. También esta mayor comunicación, entendemos, nos hará generar entregas más acorde a lo que el product owner.

3.4. Diseño Orientado a Objetos

3.4.1. Ciclo de vida de una consulta

El siguiente diagrama (figura 1) muestra la interacción en los distintos componentes de la aplicaciones.

La primera interacción es el request de la página principal index.html, que es la interfaz de usuario del prototipo, y contiene los formularios de búsqueda. El usuario hace un request tipo GET de la página, el cual es atendido por el componente Sinatra. Este último tiene una ruta registrada que indica que este tipo de request debe ser atendido directamente por el componente HAML (pasand por parámetro el nombre del recurso solicitado). El componente HAML reconstruye el contenido html en base al recurso indicado, y lo devuelve a Sinatra para ser enviado al usuario.

La segunda interacción es el request de resultados de búsqueda. El usuario hace un request tipo GET del recurso search, pasando los parámetros de búsqueda, el cual es atendido por el componente Sinatra. Este último tiene una ruta registrada que indica que este tipo de request debe ser atendido primero por el componente PrecioJustoController, el cual valida los parámetros, crea los filtros (a través de FiltersFactory), y hace el llamado al componente PrecioJustoService para procesar la búsqueda. Una vez resuelta la búsqueda, se llama al componente HAML para que construya el html con los resultados, el cual se pasa al componente Sinatra para ser enviado al usuario.

Los componentes de PrecioJustoService y de FiltesFactory serán explicados con mayor profundidad en las sucesivas secciones.

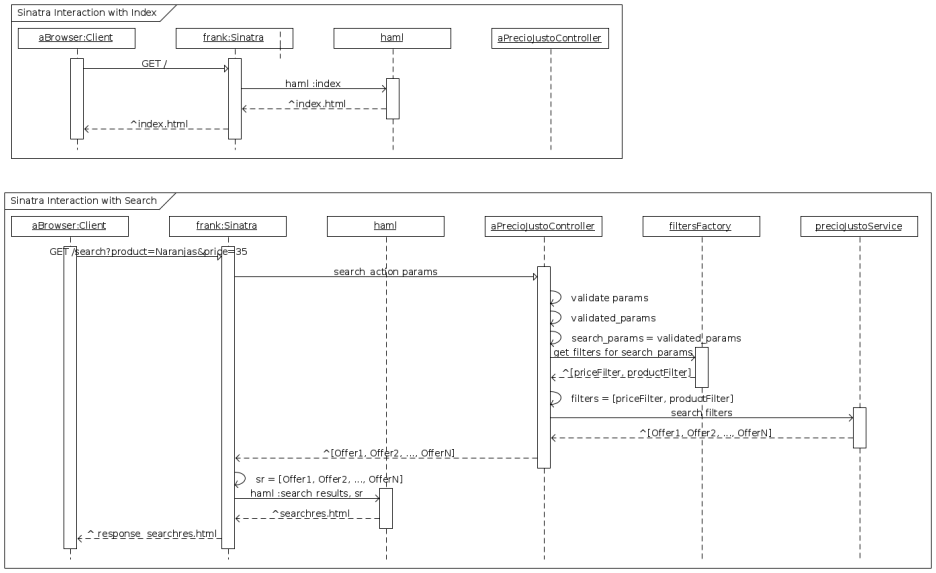


Figura 1: Diagrama de secuencia de sinatra

3.4.2. Factory de servicios

Twitter expone varias APIs para la obtención de tweets; las dos más relevantes son la API de Streaming y la API de Search. Estas dos APIs tiene comportamientos muy diferentes.

- Streaming: requiere establecer una conexión persistente HTTP (Chunked Encoding, Streaming), usando autenticación (OAuth). Devuelve los tweets que contienen los términos buscados, en tiempo real, o sea que los resultados ya devueltos no vuelven a aparecer. No impone límites a las búsquedas y cantidad de conexiones, aunque sí tiene políticas para las reconexiones.
- Search: recibe, procesa y responde cada request en su totalidad, finalizando en el request en el momento, sin requerir autenticación. Tiene un límite para la cantidad de resultados devueltos, y para la antigüedad de éstos; además impone límites para la cantidad de requests.

En el caso de la API de Streaming, una posibilidad de uso es mantener una o varias búsquedas del maestro de productos y así ir actualizando un repositorio interno/intermedio, el cual es utilizado para responder a las búsquedas de los usuarios de la aplicación.

Para la API de Search, un uso posible es resolver las búsquedas de los usuarios de la aplicación directamente contra ésta.

En nuestro caso, implementamos ambas formas de trabajar con los tweets, asegurándonos que el intercambio de una por otra sea transparente: desde el lado de clientes de la búqueda de ofertas de El Precio de Justo mantienen la misma interfaz.

Para esto último decidimos utlizar el patrón de `abstract factory` con dos implementaciones, una online (contra Twitter - API Search) y otra offline (contra un repositorio intermedio - API Streaming). Cada una de estas implementaciones nos permiten crear la clase de servicio apropiada (la clase que encapsula la lógica de la aplicación), los filtros permitidos y la implementación específica de éstos.

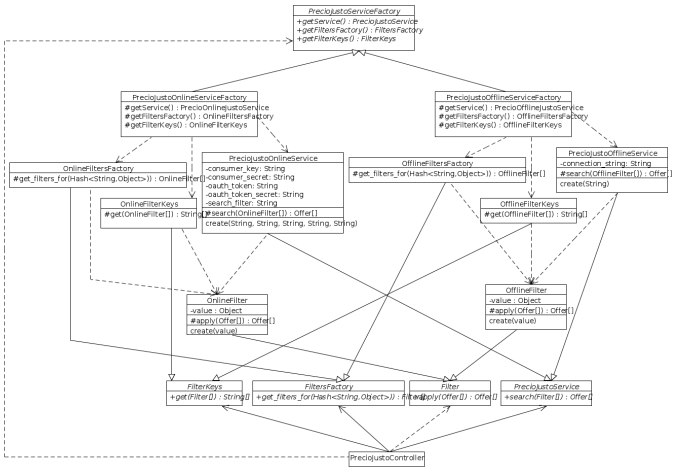


Figura 2: Diagrama de clases de creación de servicios

Esto último puede verse en la figura 2. La clase abstracta PrecioJustoServiceFactory declara los métodos getService, que devueve el tipo PrecioJustoService, getFiltersFactory, que devuelve el tipo FiltersFactory, y getFilterKeys, que devuelve el tipo FilterKeys. A su vez tenemos ambas familias de clases concretas, Online y Offline, que implementan los tipos definidos previamente.

Los tipos de FilterKeys y FiltersFactory (y Filters), son utilizados por la clase de PrecioJustoController para traducir los filtros recibidos en los requests de usuario a filtros que sabe utilizar la implementación de PrecioJustoService (este tema se verá en mayor detalle en el próximo punto).

Las clases de PrecioJustoOnlineService y PrecioJustoOfflineService implementan el tipo PrecioJustoService, y encapsulan la lógica para resolver las búsquedas (accediendo directamente a Twitter en el primer caso, y accediendo a un repositorio interno en el segundo).

3.4.3. Filtros

Dado que el filtrado de los tweets de las ofertas búscadas se realizan de distinta manera, los filtros si bien deberán proveer las misma funcionalidad son implementativamente distintos.

En un caso filtrarán las ofertas que se vayan obtienendo de twitter en vivo y en el otro caso se realizará un búsqueda sobre el motor de base de datos.

Los filtros modelados son por producto, precio y ubicación, si bien es posible extenderlos.

Esto puede verse en el diagrama de la figura 3.

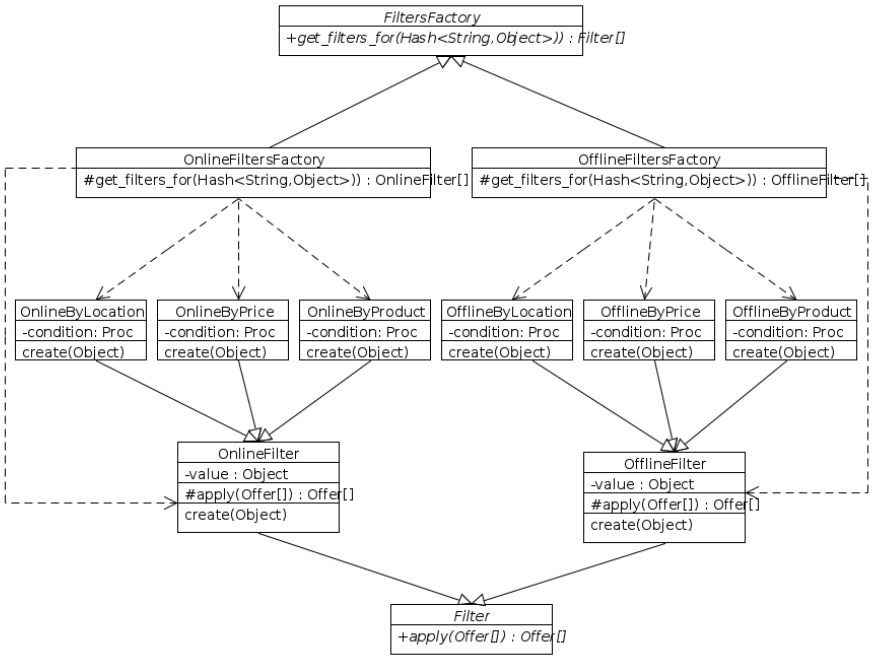


Figura 3: Diagrama de clases de creación de filtros

3.4.4. Extracción de datos de un tweet

La extracción de las información de las ofertas que se encuentra en los tweets se realiza mediante objetos de la clase `OfferFromTweetExtractor` (clase abstracta).

Para la demostración realizamos una implementación que extrae la información mediante expresiones regulares para cada pedazo de información dentro del texto, salvo para la geolocalización del tweet. Si bien no está implementado para la demo, es deseable delegar la creación de las instancias de Offer. Para ello pensamos realizar mediante un `OfferBuilder`, la tarea del mismo sería realizar las acciones necesarias para crear una nueva instanacia.

La idea es que cada pedazo de información genere una instancia de un objeto polimórfico de la información que modela (ej: el precio, producto, unidad). Cuando nos referímos a objetos polimórficos, queremos decir que la información podría no poder generar un objeto válido y queremos que resulte transparente para los objetos con los que interactuan con dicho objeto (la idea es poder implementar `Null Object Pattern`)

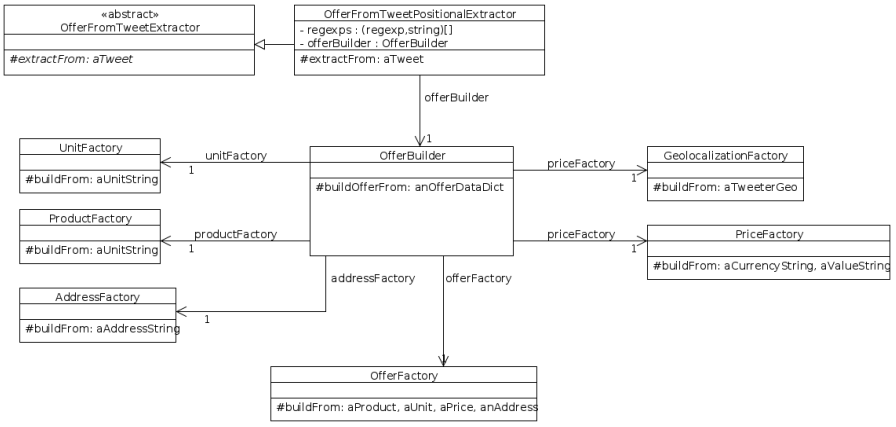


Figura 4: Diagrama de clases de extraccion datos de tweet

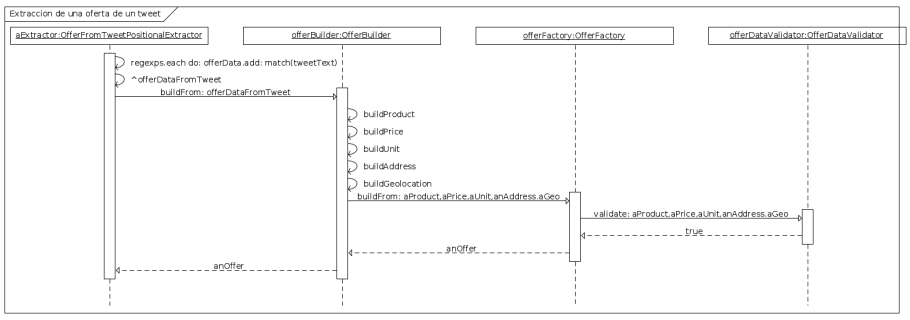


Figura 5: Diagrama de secuencia de extraccion de datos de tweet

3.4.5. Servicio Online

En este caso la clase PrecioJustoOnlineService encapsula la lógica de la aplicación utilizando como fuente de información las búsquedas directas en `Twitter` mediante la `API` de `Search`. Para aplicar los filtros utiliza las clases derivadas de `OnlineFilter` y utiliza `OfferFromTweetPositionalExtractor` para la extracción de las ofertas.

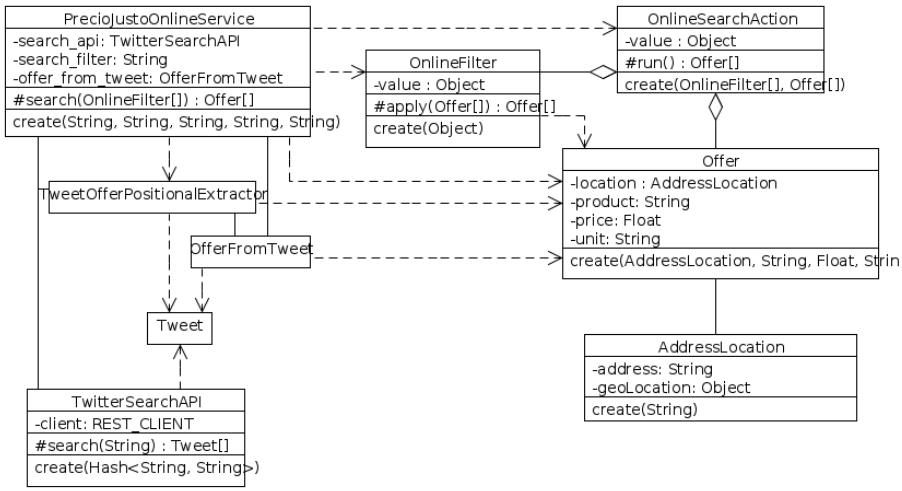


Figura 6: Diagrama de clases de servicio online

3.4.6. Servicio Offline

PrecioJustoOfflineService

En este caso la clase `PrecioJustoOfflineService` encapsula la lógica de la aplicación utilizando como fuente de información una base de datos `SQLite3`. La cual es alimentada mediante un servicio independiente.

Para aplicar los filtros utiliza las clases derivadas de `OfflineFilter`. Las ofertas ya fueron parseadas antes de ser insertadas en la base de datos.

Servicio de Streaming

Este proceso independiente utiliza la `API` de `Streaming` de `Twitter` para obtener las ofertas y persistirlas en la base de datos.

Para la extracción de las ofertas de los tweets utiliza el extractor `OfferFromTweetPositionalExtractor`.

Cliente/Servidor SQLite3

Dado la base de datos `SQLite3` es un archivo, el mismo se bloquea al accederlo directamente. Para evitar esto generamos un proceso independiente que es el único que bloque el archivo, pero expone mediante `DRB (remoting)` la conexión a esta base para poder ser usada concurrentemente por distintos proceso.

Ofertas y colaboradores

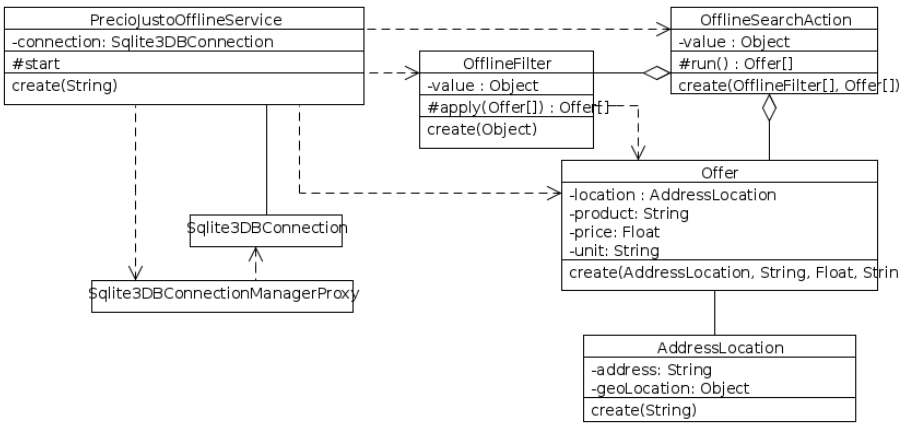


Figura 7: Diagrama de clases de servicio offline

El siguiente diagramas muestra el modelo de las ofertas (`Offer`) y sus colaboradores. Dichos colaboradores no están modelados en la demostración. El siguiente diagramas muestra la idea de creación de los distintos objetos con la idea de poder generar objetos no válidos polimórficos a los válidos.

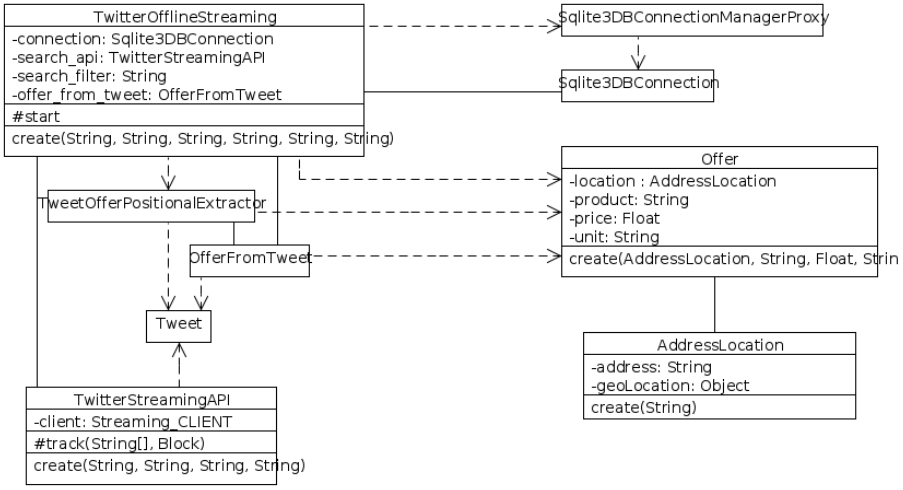


Figura 8: Diagrama de clases de streaming de twitter

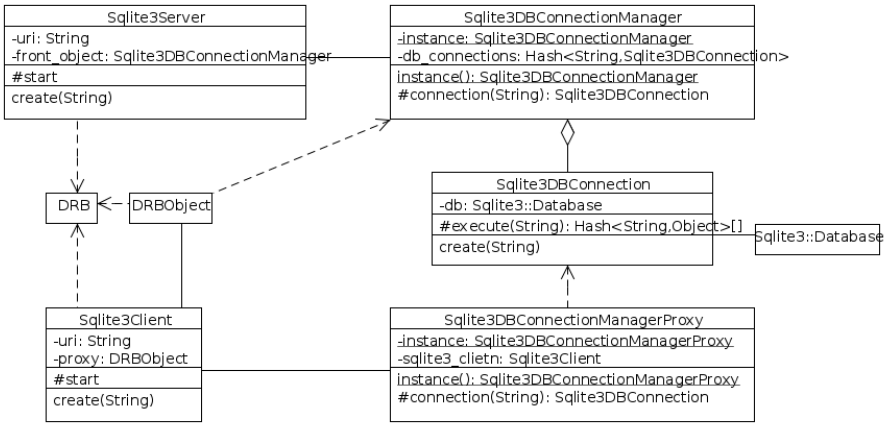


Figura 9: Diagrama de clases de BD SQLite3

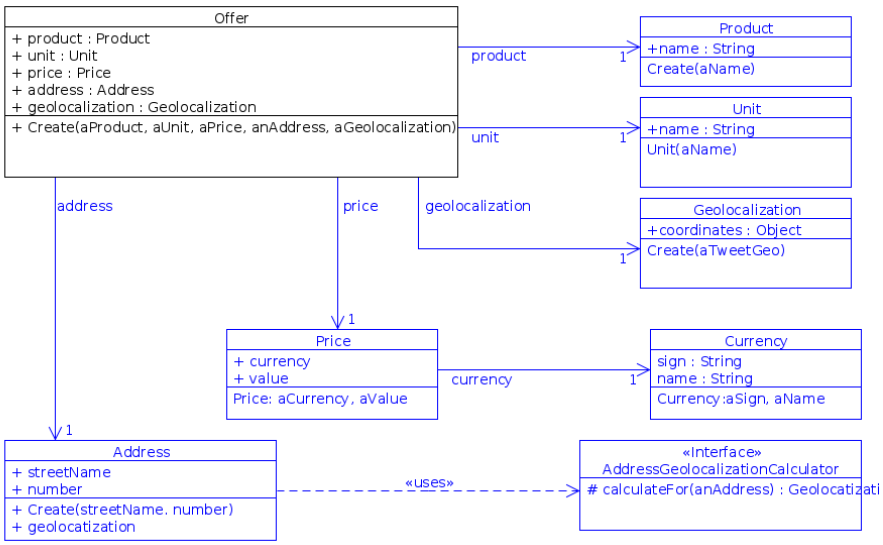


Figura 10: Diagrama de clases de offer y colaboradores

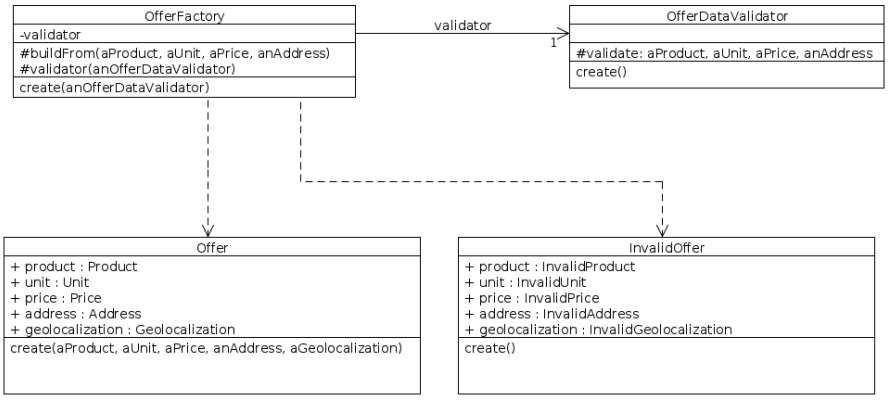


Figura 11: Diagrama de clases de creación de offer