Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e Implementación de una aplicación de escritorio para la comparativa de productos ciclistas en Tiendas Ciclistas Online

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autores: Sancho Molinero, Raúl

Tutor: Soler Bayona, José Vicente

Curso 2017 - 2018

Agradecimientos

Al equipo SAM,
por soportar mis nervios durante estos meses.

A la peña el Pedal,
por aguantar todas mis consultas y
motivarme a hacer este TFG.

Al XTension Team,
por hacer el cuestionario y siguientes consultas,
sin ellos no podría haber iniciado esto.

A mi familia,
por apoyarme en todo momento.

A Mario,
por tus buenos consejos y gran ayuda,
de forma desinteresada.

A José Vicente mi tutor,
por su gran ayuda en este TFG desde la distancia.

Muchas Gracias

Resum

Aquest projecte consistix en la creació d'una aplicació d'escriptori que compararà a temps real els productes que busque l'usuari i aconsellarà en quina web comprar preferentment. Per a la seua realització farem ús de 3 fases principals, en la primera d'elles analitzarem els requisits de l'aplicació, en la segona ens centrarem en el disseny centrat en l'usuari. Finalment, en la tercera fase realitzarem la implementació de l'aplicació fent ús de JavaFX, Jsoup i H2 Database per a la persistència de l'historial de preus i la base de dades de productes.

Paraules clau: scrapping, JavaFX, Jsoup, H2, ciclisme, fxml, Hibernate, comparador de preus.

Resumen

Este proyecto consiste en la creación de una aplicación de escritorio que comparará a tiempo real los productos que busque el usuario y aconsejará en que web comprar preferentemente. Para su realización haremos uso de 3 fases principales, en la primera de ellas analizaremos los requisitos de la aplicación, en la segunda nos centraremos en el diseño centrado en el usuario. Por último, en la tercera fase realizaremos la implementación de la aplicación haciendo uso de JavaFX, Jsoup y H2 Database para la persistencia del historial de precios y la base de datos de productos.

Palabras clave: scrapping, JavaFX, Jsoup, H2, ciclismo, fxml, Hibernate, comparador de precios.

Abstract

This project consists in the creation of a desktop application that will compare the products searched by the user in real time and advise on which web to buy preferably. For its realization we will make use of 3 main phases, in the first one we will analyze the requirements of the application, in the second we will focus on the user-centered design. Finally, in the third phase we will make the implementation of the application use of JavaFX, Jsoup and H2 Database for the persistence of the price history and the product database.

Keywords: scrapping, JavaFX, Jsoup, H2, cycling, fxml, Hibernate, price comparison.

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

En primer lugar, cabe decir que va a ser realizada una aplicación de escritorio, desarrollada con el lenguaje Java. Dicha aplicación será un comparador de precios de productos ciclistas puestos a la venta en diversas tiendas online. Para ello se hará uso de diversas librerías Java, en el caso de la interfaz gráfica se usará JavaFX, mientras que para el web scraping se hará uso de la librería JSoup.

Para la realización de esta aplicación realizaremos diversas encuestas a usuarios ciclistas que compren en tiendas tanto físicas como online. Estas se analizarán con el objetivo de hacer la aplicación al gusto de los usuarios. Tras realizar los prototipos de la interfaz realizaremos de nuevo encuestas para verificar que la interfaz no es dificultosa para el usuario final y lo mismo haremos en las diversas fases de la interfaz hasta llegar a la versión final.

1.1. Motivación

En primer lugar, cabe decir que soy ciclista de carretera y llevo mucho tiempo buscando la manera de comparar diferentes webs rápidamente, pero no hay muchas webs o aplicaciones específicas que facilite esta tarea. Por este motivo he pensado en realizar esta aplicación, además no solo me facilitaría a mí la búsqueda de productos, sino que también podría ayudar a toda la comunidad ciclista, debido a que hay muchas tiendas online y se hace complicado poder comparar todas las webs rápidamente.

Además, hoy el comercio electrónico está teniendo una tendencia al alza muy elevada y los usuarios buscan ante esta gran oferta de diversas tiendas el mejor precio [1], y se intenta rascar hasta el último céntimo, es por ello que hoy en día cada vez más están apareciendo más comparadores de precio, con el objetivo de que el usuario tenga una mayor facilidad a la hora de encontrar el mejor precio.

Por lo anteriormente expuesto voy a realizar este Trabajo de Fin de Grado (TFG) con los objetivos que serán descritos a continuación.

1.2. Objetivos

La aplicación desarrollada en este TFG está dirigido a usuarios que sean ciclistas de carretera o de montaña y que sean asiduos a tiendas online, y ante la problemática expuesta en el punto anterior la aplicación desarrollada en el presente TFG deberá cumplir con los siguientes objetivos:

- Realizar una aplicación de escritorio que compare el precio del producto que elija el usuario entre los que haya en un listado, entre más de 5 tiendas online distintas.
- Hacer que esta aplicación le muestre al usuario una gráfica del progreso de los precios de forma histórica y hacer que el usuario también pueda escoger

las tiendas entre las que quiere filtrar dentro de un listado de tiendas soportadas.

1.3. Plan de trabajo

Con el objetivo de planificar el trascurso del TFG se han identificado las diversas tareas a realizar para conseguir con éxito los objetivos planteados, dichas tareas se muestran en la figura 1.1.

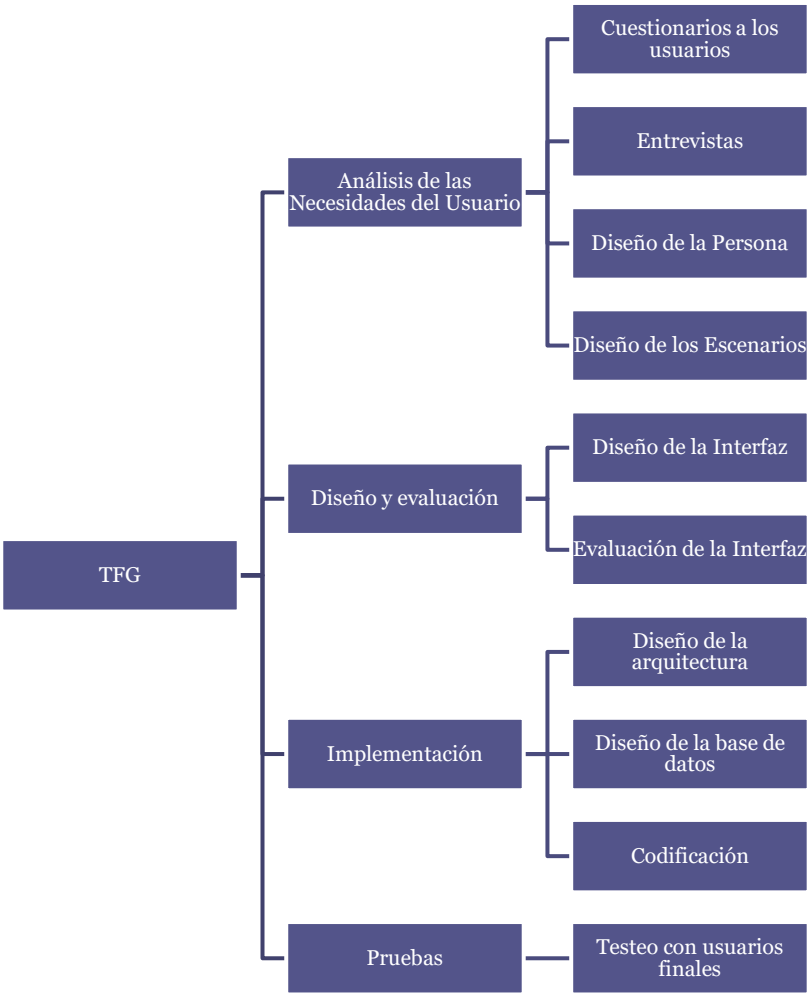

Figura 1.1 Esquema de plan de trabajo

Las tareas expuestas en la figura 1.1, engloban todo el índice de contenido desde el punto 2 y ha sido elegido de esta forma porque a la hora de distribuirme las tareas a lo largo del tiempo en el que he realizado este TFG me es más sencillo hacerlo de esta manera y además de esta forma sigo el estilo de Diseño Centrado en el Usuario haciendo las fases de Análisis de las necesidades, diseño y evaluación y la última fase de implementación.

1.4. Estructura de la memoria.

A lo largo de este documento se va a exponer en primer lugar el contexto tecnológico en el que nos encontramos, explicando todo el material que hemos utilizado como librerías y lenguajes utilizados en el transcurso del TFG. Seguidamente se pasa a analizar las necesidades del usuario donde se realizarán unos cuestionarios a los usuarios, con estos se diseñarán a la persona principal y persona secundaria y seguidamente se expondrán los diferentes escenarios. A continuación, se comentará el diseño de la aplicación comentando los prototipos diseñados, su evaluación y seguidamente la presentación del diseño final de la interfaz teniendo en cuenta las evaluaciones. Tras realizar el análisis se pasa a explicar cómo se ha realizado la implementación comentando las distintas capas utilizadas, pasando por la capa lógica, la capa de presentación y por último la capa de persistencia. Seguidamente se harán unos ejemplos de uso en los que se mostrará como el usuario hace uso de la aplicación. Y por último se expondrá la conclusión con posibles mejoras a futuro de la aplicación.

2. Contexto Tecnológico

En este apartado os vamos a comentar cual es la situación tecnológica ante la que nos encontramos en cuanto a que aplicaciones ya existen que hagan funciones parecidas, haremos una breve critica a las más importantes y se presentará la propuesta de mejora, y seguidamente os explicaremos cual es el entorno de desarrollo elegido, las principales librerías utilizadas y los componentes utilizados en el TFG.

2.1. Aplicaciones existentes

En la actualidad existen muchos comparadores de precios, pero de normal todos suelen ser para vuelos, seguros, ropa, etc. Como las siguientes aplicaciones:

- Kayak [2]: Es una web que hace cientos de búsquedas diarias para encontrar el mejor precio en hoteles, vuelos, coches de alquiler o paquetes vacacionales.
- Skyscanner [3]: Es una aplicación web destinada a hacer cientos de búsquedas diarias para encontrar los mejores precios en vuelos principalmente.
- Rastreator [4]: El objetivo de esta web es la comparación de seguros de todo tipo para dar al usuario la compañía de seguros con el mejor precio.
- Google Shooping [5]: Google creó esta aplicación para dar al usuario el mejor precio para cualquier producto con el objetivo de hacer que el usuario se ahorre dinero.

Los comparadores de precio anteriormente comentados son 3 de los comparadores de precio más usados por el usuario, pero todos ellos son destinados a viajes o a seguros, y la última de ellas sí que puede usarse para productos ciclistas, pero no está 100% especializada en esa área. Por otro lado, existen como plataformas para comparación de precios para productos ciclistas las siguientes aplicaciones:

- CoreBicycle [6]
- Comparaciclismo [7]
- NBICI [8]

Estas tres aplicaciones están destinadas 100% a usuarios ciclistas, comparando todo tipo de productos ciclistas, todas ellas comparando algunas de las principales webs como son Chain Reaction o Retto.

2.2. Critica a las aplicaciones existentes

Como hemos visto anteriormente actualmente existen 3 aplicaciones destinadas a ciclistas, pero la única que tiene un diseño amigable es CoreBicycle, ya que las otras 2 aplicaciones aun parece que no estén totalmente en la fase de producción, sino que aun estén en la fase de desarrollo.

Además, casi todas ellas no pasan de 5 webs en las que busquen el producto deseado por el ciclista, siendo este número de tiendas pequeño. Como adición el usuario solo

puede saber el estado actual del precio, pero no puede ver la tendencia que tiene el precio en la actualidad ya sea alcista o bajista.

2.3. Propuesta

Tras lo anteriormente expuesto hemos decidido realizar "Bikestreator" una aplicación destinada a la comparación de precios ciclistas con el objetivo de mejorar la aportación a la comunidad ciclista de esas 3 aplicaciones, planteando las siguientes mejoras principales:

- Más de 6 tiendas donde comparar.
- Hacer que el usuario pueda ver una gráfica que indique la tendencia del precio.

Para ello haremos uso de un entorno de programación y unas librerías que se explicarán a continuación.

2.3.1. Entorno

Bikestreator, que es como llamaremos a nuestra aplicación, ha sido desarrollada como una aplicación de escritorio para su ejecución en sistemas operativos como Windows, Ubuntu, Debian y MacOS, siempre y cuando tengan instalada la JVM de Java [9].

Para el desarrollo de la aplicación hemos elegido utilizar el entorno de programación Eclipse IDE [10], es una herramienta que originalmente desarrollada por la multinacional International Business Machines (IBM) y posteriormente fue liberada y empezó a ser desarrollada por Eclipse Foundation. Además, es un entorno de desarrollo integrado para usarse de forma modular, teniendo módulos como Java, C, C++ y PHP, en nuestro caso hemos utilizado el módulo de Java.

Por otro lado, cabe decir que también existen otros entornos de desarrollo para Java entre los que se encuentra NetBeans, entorno desarrollado inicialmente por Sun y actualmente por Oracle, esta herramienta es una de las principales a nivel mundial junto a eclipse.

2.3.2. Librerías

Este trabajo de fin de grado va a ser desarrollado en Java y Java FX [11], siendo JavaFX el lenguaje utilizado para la interfaz gráfica, debido a que se trata de un lenguaje destinado a crear aplicaciones de escritorio, para teléfonos móviles o para webs.

Por otro lado, hemos utilizado para la base de datos:

- **Hibernate** [12]: Es una herramienta de Mapeo objeto-relacional (Object-Relational mapping, ORM) que nos ayuda a mapear los objetos entre un lenguaje de programación orientado a objetos, como es nuestro caso con Java, y la base de datos.

- **H2** [13]: Es la base de datos relacional programada en Java y podemos lanzar peticiones SQL sin la necesidad de conexiones engorrosas como si sucede con otros tipos de bases de datos relacionales como MySQL o SQL Oracle.

En cuanto a la recopilación de datos de las webs que vamos a inspeccionar haremos uso de JSoup [14], que es una librería Java de código abierto para la extracción de información desde HTML o CSS.

3. Análisis de las necesidades del usuario

En este apartado vamos a analizar las necesidades del usuario, que es una de las fases del Diseño Centrado en el Usuario (DCU) [15], dentro de esta fase analizaremos que necesita realmente el usuario, y para ello hemos realizado a diversos usuarios ciclistas un cuestionario, en el cual se les ha preguntado los siguientes puntos:

- Edad
- Ocupación
- Cuál es su dominio de las nuevas tecnologías.
- Si practica cualquier otro deporte a parte del ciclismo.
- Como se definiría el usuario frente a unas formas de llamarse los ciclistas de forma coloquial.
- Que modalidad de ciclismo practican.
- Acerca del tiempo que llevan practicando ciclismo.
- La frecuencia con la que suelen practicar dicho deporte.
- Acerca de donde suelen comprar si en tiendas físicas u online.
- Acerca de en qué webs suelen comprar.
- Que suelen comprar en las webs.

3.1. Cuestionario

En el siguiente a apartado mostraré cual es el cuestionario completo que les hemos realizado que podemos apreciar en la figura 3.1, y posteriormente hemos pasado a analizar los resultados de dichos cuestionarios, como podemos ver en el siguiente punto.

Encuesta TFG Aplicación Ciclista

Esta encuensta se hace con el objetivo de recavar información de usuarios que sean ciclistas y hagan compras de productos cidlistas via internet.

*Obligatorio

Edad *

Tu respuesta

Ocupación *

- [ ] Estudiante
- [ ] Ama/o de casa
- [ ] Desempleado
- [ ] Trabajo Estable
- [ ] Trabajo Temporal
- [ ] Jubilado

Nivel de Estudios *

- ○ E.S.O.
- ○ BAchillerato
- ○ F.P. Medio
- ○ F.P. Superior
- ○ Formación Universitartia
- ○ Master o Doctorado

Indica tu nivel de destreza en las nuevas tecnologías *

	0	1	2	3	4	5	
Nulo	○	○	○	○	○	○	Experto

¿Como te definirías? *

- ○ Ciclista
- ○ Globero
- ○ Persona Sedentaria
- ○ Otro: _____

¿Practica aún otro deporte a parte de ciclismo? indique cuales *

- [ ] Running
- [ ] Natación
- [ ] Ningún otro deporte
- [ ] Otro: _____

SIGUIENTE

Datos Especificos

En este apartado buscamos saber vuestras preferencias cuando comprais productos ciclistas.

¿Que modalidad de ciclismo realiza? *

- [ ] MTB
- [ ] Carretera
- [ ] BMX
- [ ] Trial
- [ ] Otro: _____

¿Cuanto tiempo llevas practicando ciclismo? *

- ○ < 6 meses
- ○ < 1 año
- ○ < 2 años
- ○ > 2 años

¿ Con que frecuencia sales a rodar en bicicleta a lo largo de la semana? *

- ○ 0 días
- ○ 1 día
- ○ 2 días
- ○ 3 días
- ○ 4 días
- ○ 5 o más veces

¿Donde sueles comprar artículos ciclistas? *

- [ ] Tienda Física
- [ ] Tienda Online/Web

En caso de haber respondido "Tienda Online/web" en la pregunta anterior indique en cual/es *

- [ ] BikeShop
- [ ] Alltricks
- [ ] Chain Reaction
- [ ] Retto
- [ ] BikeInn
- [ ] BiciMarket
- [ ] Deporvillage
- [ ] Otro: _____

¿Que sueles comprar en las tiendas online? *

- [ ] Componentes
- [ ] Ropa
- [ ] Accesorios
- [ ] Herramientas
- [ ] Bicicletas
- [ ] Otro: _____

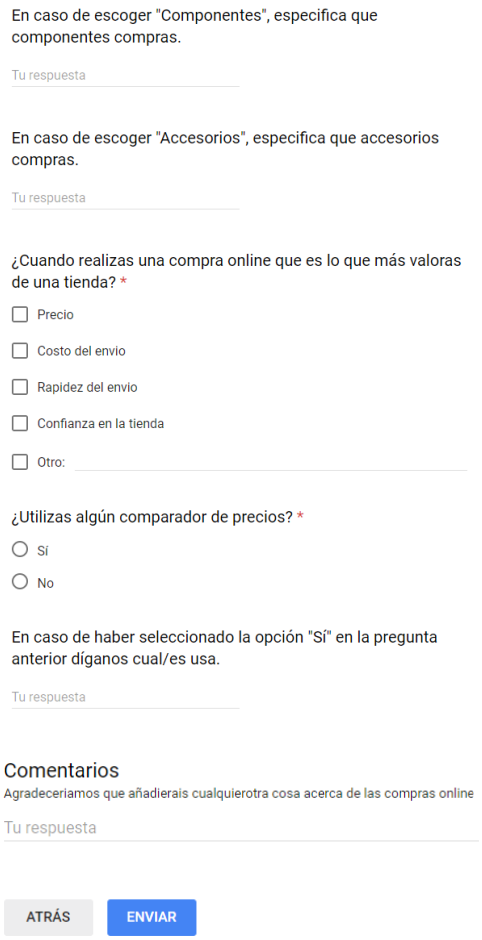

Figura 3.1 Cuestionario Análisis de Necesidades del Usuario

Para la realización de este formulario hemos hecho uso de la plataforma Google Forms [16], con la que hemos desplegado el cuestionario a los usuarios y posteriormente hemos extraído con dicha herramienta todos los datos en Excel para su análisis.

3.1.1. Análisis del cuestionario

En este apartado vamos a realizar el análisis de las respuestas que hemos obtenido al enviar la encuesta a una peña ciclista y que sus integrantes nos hayan respondido, en total han respondido a la encuesta un total de 28 ciclistas. Para ello primero mostrare una tabla resumen en la cual podremos ver un resumen de las 28 respuestas a la encuesta en la tabla 1.

Total de encuestados	28
Media de Edad	39,4
Ocupación	
Estudiante	6
Ama/o de Casa	1
Desempleado	3
Trabajo Estable	17

Trabajo Temporal	3
Jubilado	1
Nivel de Estudios	
E.S.O	0
Bachillerato	3
F.P. Medio	1
F.P. Superior	3
Formación Universitaria	13
Máster o doctorado	8
Nivel de destreza en las nuevas tecnologías	
0	0
1	0
2	2
3	9
4	11
5	6
¿Cómo te definirías?	
Ciclista	21
Globero	6
Persona sedentaria	0
Triatleta	1
¿Practica algún otro deporte?	
Running	11
Natación	9
Ninguno mas	10
Senderismo	1
Gimnasio	5
Triatlón	1
Montaña	1
Modalidad de ciclismo practicada	
MTB	7
Carretera	28
BMX	0
Trail	0
Pista	1
Tiempo practicando ciclismo	
<6 meses	0

<1 año	1
<2 años	1
>2 años	26
Frecuencia con la que se entrena	
0 días	0
1 día	7
2 días	12
3 días	5
4 días	1
5 días o mas	3
Donde suelen comprar artículos ciclistas	
Tienda Física	18
Tienda Online	23
Webs en las que compran	
BikeShop	7
Alltricks	6
Chain Reaction	13
Retto	1
BikeInn	6
BiciMarket	4
Deporvillage	12
Fabregues bicicletas	2
Wiggle	1
Amazon	2
Valwynd cycles	1
Mantel	1
AliExpress	2
Que suelen comprar	
Componentes	13
Ropa	16
Accesorios	17
Herramientas	7
Bicicletas	6
No compro Online	1
Qué valoran al comprar en una web	
Precio	25
Costo del envío	7

Rapidez del envío	12
Confianza en la tienda	11
Calidad-Precio	1
No compro Online	1
Utilizas algún Comparador	
Sí	1
No	27

Tabla 3.1 Resumen de Resultados

Como podemos ver en la tabla 3.1, 25 de los ciclistas encuestados lo que más valoran al comprar en una web es su precio y que 27 de los encuestados no usan ningún comparador de precios y esto es debido a que no hay ningún comparador específico para productos ciclistas, es por ello que se puede considerar necesaria nuestra aplicación para los usuarios potenciales debido a que hay muchísimas tiendas online diferentes.

Tras analizar los datos también nos hemos dado cuenta de que los usuarios potenciales hacen uso de 3 tiendas más de las que habían sido propuestas es por ello que Fabregues Bicicletas, Mantel y Wiggle, serán añadidas a las tiendas online que estarán en nuestra aplicación. Por otro lado, también cabe destacar que los 28 encuestados realizan ciclismo de carretera por lo que la aplicación se centrará en ciclismo de carretera, pero 7 de ellos también realizan MTB, por lo que también estarán disponibles algunos productos que sean de MTB, pero en su mayoría la aplicación estará enfocada a ciclistas de carretera.

Con el objetivo de realizar posteriormente a las personas hemos obtenido como puntos clave:

- Mayoritariamente los usuarios que harán uso de la aplicación tendrán formación universitaria.
- Tienen conocimiento medio de la tecnología, por lo que la aplicación no tendrá un diseño muy minimalista.
- Edad media 39.5, pero con usuarios muy jóvenes.
- Compran en su mayoría componentes, ropa y bicicletas de carretera.

3.2. Entrevistas

En este apartado lo que vamos a realizar es una entrevista a 2 usuarios potenciales, con el objetivo de entrar más en detalle y de esta forma poder realizar un “Usuario”. Para ello vamos a realizar una serie de preguntas a nuestros usuarios entrevistados, las cuales serán las siguientes:

- ¿Qué tipo de accesorios sueles comprar y en que te sueles fijar más?
- ¿Qué estilo de ropa ciclista te gusta más?
- ¿Qué componentes sueles cambiar a la bicicleta y por qué?
- ¿Qué esperas de una aplicación que compare precios?

A continuación, mostraremos la entrevista realizada a los dos usuarios escogidos y con ello podremos realizar los usuarios.

3.2.1. Entrevista usuario 1

P. ¿Qué tipo de accesorios sueles comprar y en que te sueles fijar más?

R. Cuando miro accesorios, lo que más me gusta mirar suelen ser cascos y gafas, aunque también GPS

P. ¿Qué estilo de ropa ciclista te gusta más?

R. En general toda, aunque ahora miro mucho las gorras, los calcetines, así como maillots y culotes cortos

P. ¿Qué componentes sueles cambiar a la bicicleta y por qué?

R. Lo último que cambié fue el sillín, aunque pronto he de cambiar las cubiertas y la cinta de manillar. Lo que cambio normalmente es por desgaste

P. ¿Qué esperas de una aplicación que compare precios?

R, Que a diferencia de otras pueda tener un gran abanico de páginas y ofertas, para así poder obtener rápidamente cual es el mejor precio del mercado.

3.2.2. Entrevista usuario 2

P. ¿Qué tipo de accesorios sueles comprar y en que te sueles fijar más?

R. Cubiertas o ropa, en la calidad y durabilidad.

P. ¿Qué estilo de ropa ciclista te gusta más?

R. La que tenga una mayor flexibilidad

P. ¿Qué componentes sueles cambiar a la bicicleta y por qué?

R. Cubiertas, por el desgaste

P. ¿Qué esperas de una aplicación que compare precios?

R, Conseguir un mayor ahorro en aquello que compre.

3.3. Persona

Tras realizar las encuestas y las entrevistas a los usuarios potenciales y analizar los datos obtenidos podemos pasar a diseñar a la persona primaria y secundaria que usaremos a lo largo del desarrollo de la aplicación, que coinciden con el usuario 1 y 2 de las entrevistas respectivamente.

3.3.1. Persona Primaría

En este apartado vamos a mostrar a la persona primaría que hemos realizado tras analizar los datos de la entrevista del usuario 1 y los datos de los cuestionarios.

Nombre: Adrián Contador Indurain
Edad: 34 años
Ocupación: Estudiante de Oposición.

Biografía

- Vive con su pareja en un piso situado en Llombai.
- Está estudiando las oposiciones de Magisterio.
- Tiene un gran nivel de conocimiento en las nuevas tecnologías.
- Está especializado en Magisterio en Educación Física.

Deporte y preferencias de compra

- Desde siempre ha amado el deporte, pero desde hace 6 años hace ciclismo.
- Se inició con el ciclismo de montaña y más tarde se pasó al ciclismo de carretera.
- Le gusta cuidar mucho su bicicleta y por ello constantemente está buscando nuevos productos y accesorios para ella.
- Lo que más compra son neumáticos para la bicicleta y siempre busca en diversas webs.
- Es amante de las gorras de bici y calcetines extravagantes.

Objetivos

- Está buscando formas para encontrar el mejor precio de los productos que más suele comprar.
- Busca una aplicación que cuando el haga la búsqueda dicha web no busque entre pocas webs, sino que use un gran abanico de tiendas online.

3.3.2. Persona Secundaria

A continuación, mostraremos a la persona secundaria que hemos diseñado para nuestro caso, habiendo analizado la entrevista del segundo usuario y los datos de los cuestionarios

Nombre: Alberto San Juan Linares
Edad: 32 años
Ocupación: Trabajo estable

Biografía

- Vive con sus padres en una casa adosada en Sevilla.
- Actualmente trabaja como diseñador gráfico.
- Domina las nuevas tecnologías, gracias a su formación como diseñador gráfico.

Deportes y preferencias de compra

- Lleva practicando ciclismo desde que tenía 10 años.
- Primero se inició con el mountain bike (MTB), pero a los 18 se decidió por cambiar a ciclismo de carretera y desde entonces practica las dos modalidades, pero en su gran mayoría ciclismo de carretera.
- Es un amante de las llantas y los neumáticos, por lo que constantemente está buscando comprar alguno nuevo.
- Siempre que compra algo por internet se fija primero en los costes de envío, si son gratuitos o no.
- Cuando practica ciclismo de carretera prefiere usar la ropa más ajustada y duradera que pueda encontrar, pero en MTB le gusta que sea holgada.

Objetivos

- Encontrar los neumáticos que más le gustan al mejor precio.
- Quiere que haya una gran batería de tiendas y poder encontrar el mejor precio en el mercado.

3.4. Escenarios

En este apartado vamos a realizar los diversos escenarios que se van a desarrollar a lo largo del uso de la aplicación. Un escenario es la descripción del diseño desde el punto de vista de una persona específica, indicando como esta persona alcanza sus objetivos usando la aplicación. En nuestro caso será Adrián la persona que usaremos para los escenarios, debido a que es la persona primaria.

Vamos a realizar diversos escenarios que diferenciaremos entre generales y específicos.

3.4.1. Escenarios Generales

- **Buscar el precio de Ciclismo de un componente**

Adrián llega a casa después de haber salido a entrenar 6h con su bicicleta Merida Reacto 4000 por la zona de Cullera, pero ha rajado el neumático delantero, llegando a casa, y necesita un nuevo juego de neumáticos, ya que ya tenía muy gastada el neumático trasero y aprovecha esta compra para comprar un nuevo juego de neumáticos. Entonces abre la aplicación en su PC y escoge la opción “Componentes” y sigue los pasos para buscar sus neumáticos Continental Grand Prix 4000 SII.

- **Buscar el precio de Ciclismo de ropa**

Adrián lleva varios días pensando en comprarse unos calcetines nuevos muy extravagantes, de colores llamativos y estampados aún más llamativos. Para ello, cuando llega de clase enciende el ordenador y abre la aplicación y hace click sobre la opción “Ropa”.

3.4.2. Escenarios Específicos

- **Buscar el precio de Ciclismo de Carretera**

Adrián llega a casa después de haber salido a entrenar 6h con su bicicleta Merida Reacto 4000 por la zona de Cullera, pero ha rajado el neumático delantero, llegando a casa, y necesita un nuevo juego de neumáticos, ya que ya tenía muy gastada el neumático trasero y aprovecha esta compra para comprar un nuevo juego de neumáticos. Entonces abre la aplicación en su PC y escoge la opción “Compras Carretera”. A continuación, se le actualiza la ventana de la aplicación y le aparecen 3 desplegables con los siguientes mensaajes: categoría, Marca, Modelo.

Hace click sobre el desplegable de “Categoría” y seguidamente el escoge la categoría “Neumaticos y ruedas”. Tras hacer esto, Adrián piensa en qué modelo de neumaticos quiere, ya que está indeciso entre los Continental Grand Prix SII o los Michelin Pro, finalmente se decide por los Continental, y por lo tanto hace click en el desplegable de “Marcas” y escoge la marca Continental y seguidamente escoge el modelo deseado en el desplegable de modelos.

Tras realizar la selección del modelo Adrián hace click sobre el botón de "Comparar" y seguidamente la aplicación le muestra las tiendas en orden creciente por precio. Adrián ve que en la web Chain Reaction el neumático que él quiere está a 10€ más barato que en la segunda tienda más barata, por lo tanto, escoge esta tienda para su compra y hace click sobre el botón comprar y es redirigido a la web de la tienda online.

- **Buscar el precio de Ciclismo de Montaña**

Adrián lleva varios días pensando en comprarse unos calcetines nuevos muy extravagantes, de colores llamativos y estampados aún más llamativos. Para ello, cuando llega de clase enciende el ordenador, abre la aplicación y hace click sobre la opción "Ropa" del desplegable y seguidamente elige en el desplegable de "Categorías" y escoge "Ropa" y seguidamente en la pestaña de marca escoge su marca favorita de calcetines y en el desplegable de "Tipo" escoge "Calcetines".

Adrián sabe que quiere unos específicos que ha visto en una revista online y en el desplegable de modelo escoge el modelo que ha visto en la revista.

Seguidamente la aplicación le muestra las diversas webs en las que está disponible dicho producto y los ordena de mejor a peor precio. Adrián ve que en la web Retto es la web que más barato lo tiene, como esta web es una web donde él ya ha comprado varias veces y es de confianza, le da al botón de comprar y es redirigido a la web de compra de Retto.

4. Diseño y Evaluación Iterativos

En este apartado os explicaremos como hemos realizado el diseño de la interfaz de usuario y su posterior evaluación y los cambios que hemos realizado sobre los mismos teniendo en cuenta la opinión de los usuarios que tras realizar los prototipos entrevistaremos.

4.1. Diseño de la interfaz

En este apartado vamos a presentar el prototipo realizado para la aplicación, y que posteriormente será modificado tras la evaluación.

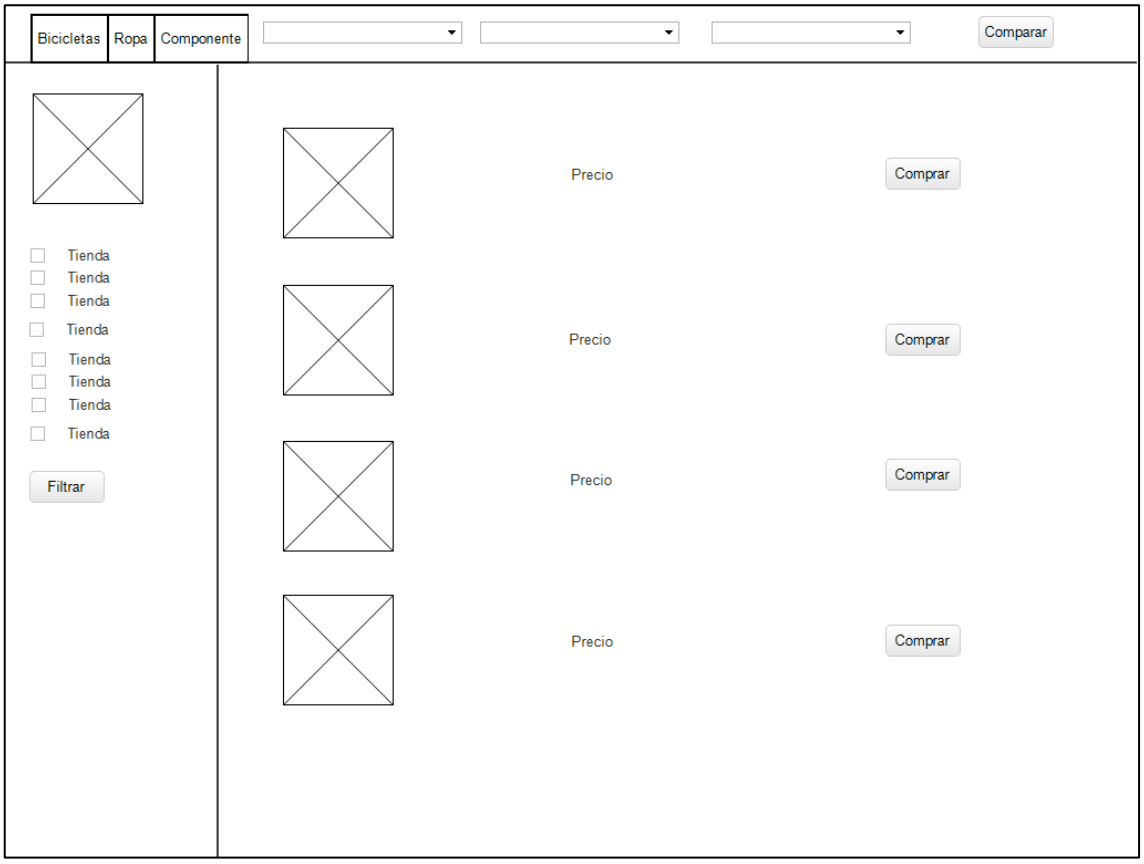

Figura 4.1 Pantalla Inicial

Tras realizar este prototipo hemos pasado a aplicar las leyes de Gestalt [17]. Los psicólogos de Gestalt realizaron una serie de leyes de percepción para la mejora de la percepción y nosotros usaremos varias de ellas para mejorar la percepción de la interfaz por parte del usuario como son las leyes de: proximidad y clausura. Por este motivo ha sido modificado el prototipo de la interfaz como hemos podido ver en la figura 4.2.

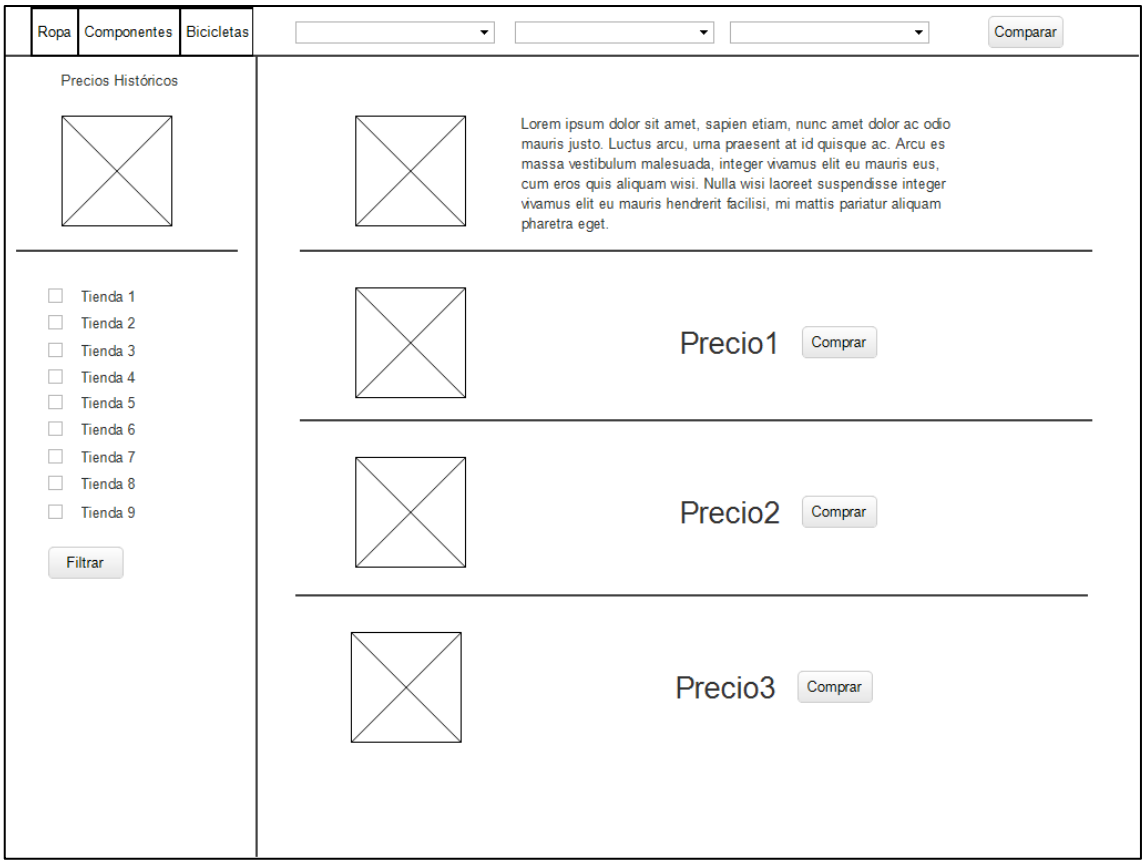

Figura 4.2 Prototipo con leyes de Gestalt

La figura 4.2. es el prototipo de la aplicación donde podemos ver que hay 3 pestañas, una pestaña por cada una de las categorías de productos ofrecidos en esta aplicación. Por lo que el flujograma para comparar sería el siguiente:

Figura 4.3 Flujograma de comparación

En cuanto a la filtración por tienda tendremos que seguir el flujo de la comparación y seguidamente realizar el siguiente flujo:

Figura 4.4 Flujograma Filtración

4.2. Evaluación de la Interfaz

Tras elaborar este prototipo pasaremos a evaluarlo mediante testeos con usuarios finales que sean ciclistas y que posteriormente la usarían. Para ello escogeremos 2 ciclistas de una peña ciclista y les realizaremos un cuestionario a cada uno de los usuarios.

El realizar esta evaluación tiene como objetivo encontrar posibles errores en la interfaz que afecten en la usabilidad de la aplicación y de esta forma solucionarlos antes de pasar a la fase de implementación.

4.2.1. Cuestionario

A continuación, vamos a realizar el cuestionario que posteriormente presentaremos a nuestros usuarios entrevistados y para ellos les realizaremos el siguiente cuestionario:

- ¿Le resulta sencilla la selección del producto?
- ¿Qué opina a cerca del sistema usado para la selección del producto?
- ¿Qué cambiaría de dicho proceso?
- ¿Cree que es sencillo encontrar el mejor precio con nuestra aplicación?

Seguidamente hemos realizado este cuestionario a nuestro usuario principal y el resultado de la entrevista ha sido el siguiente:

Pregunta: ¿Le resulta sencilla la selección del producto?

Respuesta: Me resultaría más sencillo si tuviera todo en una misma ventana y en caso de querer acceder a comparar una prenda de ropa y seguidamente un componente no tener que cambiar de pestaña en la aplicación.

P: ¿Qué opina a cerca del sistema usado para la selección del producto?

R: Me ha gustado bastante el sistema de desplegables para seleccionar el producto que busco.

P: ¿Qué cambiaría de dicho proceso?

R: Lo anteriormente dicho en la pregunta de si me resulta sencilla la selección del producto.

P: ¿Cree que es sencillo encontrar el mejor precio con nuestra aplicación?

R: Es muy sencillo, incluso me gusta el echo de que la aplicación me muestre una gráfica del precio histórico de dicho producto.

4.2.2. Conclusiones tras realizar los cuestionarios

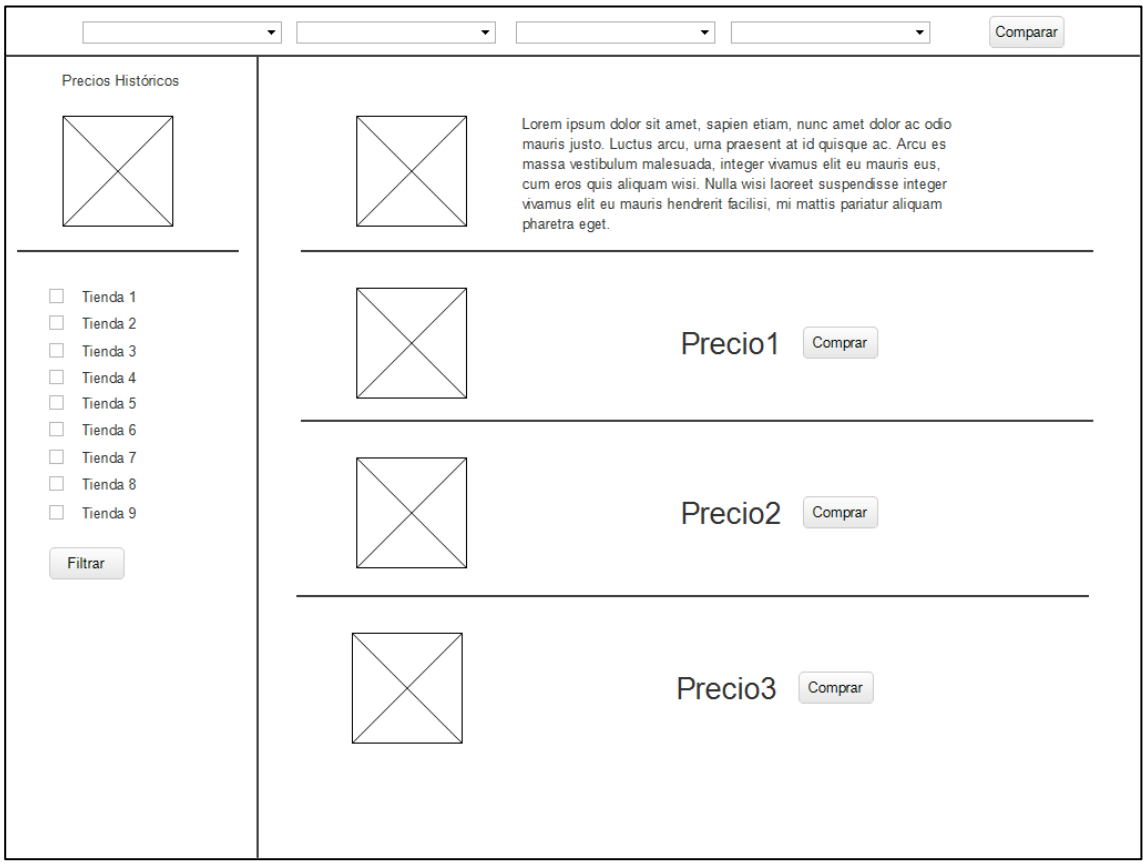

Figura 4.5 Prototipo final

Tras analizar las respuestas de nuestro entrevistado hemos llegado a la conclusión de que les gustaría más con el modelo de una única página e interactuar en ella, pudiendo cambiar de producto entre todas las opciones fácilmente.

5. Implementación

En este apartado vamos a explicar cómo hemos realizado todo el proceso de implementación desde el análisis de las webs para realizar el scrapping hasta la explicación del código Java utilizado en la aplicación.

5.1. Arquitectura

Para esta aplicación vamos a hacer uso de la arquitectura en tres capas [18]: capa de presentación, capa lógica y capa de persistencia.

La capa de presentación es aquella capa en la que ve el usuario que use la aplicación y por lo tanto en nuestro caso será la carpeta "Vistas" de nuestro proyecto y en el punto 4.2.2 de esta memoria podemos ver cuáles son las vistas utilizadas en esta capa.

En cuanto a la capa lógica es aquella capa que contiene todas las clases que implementan todas las peticiones que el usuario realizará al usar la aplicación y además será la capa que comunique la capa presentación con la capa de persistencia, que en nuestro caso la capa lógica esta completada por la carpeta del proyecto "Algoritmos".

En cuanto a la capa de persistencia podemos decir que es la capa en la que se encuentra la base de datos de la aplicación y es donde se guardan en nuestro caso los precios registrados en cada búsqueda y la información relativa a los productos para su posible búsqueda por parte del usuario.

5.2. Capas Implementadas

En este apartado os vamos a explicar cómo hemos implementado cada una de las diferentes capas usadas.

5.2.1. Capa lógica

A continuación, vamos a explicar la capa lógica que hemos utilizado en la implementación de este proyecto.

Comparar Producto

El usuario tiene que escoger una categoría de producto, marca, tipo y modelo del producto, y una vez a escogido el producto al darle al botón *Comparar*, se llama al método *comparar()* y se comparan los productos.

En primer lugar, se comprueba que el usuario haya completado correctamente los campos necesarios como podemos ver en la figura 5.1.

```
String mensajeError ="";
if(marca.getValue()==null) mensajeError+="Debeseleccionar una marca de producto. \n";
if(tipo.getValue()==null) mensajeError+="Debe seleccionar un tipo de producto. \n";
if(modelo.getValue()==null) mensajeError+="Debe seleccionar un modelo de producto.";
if(mensajeError.length() >0){
    Alert alerta = new Alert(AlertType.ERROR);
    alerta.setTitle("Error en la selección");
    alerta.setContentText(mensajeError);
    alerta.showAndWait();
}
```

Figura 5.1 Comprobación de campos

Seguidamente se hace un listado de tipo *Producto* con el precio del producto por tienda, llamando al método *getListmejorPrecio(nombreProducto),* que como podemos ver en la figura 5.2, este a su vez llama a cada uno de los métodos de scrapping realizados para cada una de las webs, que más adelante explicaremos. Además podemos ver en la figura 5.2 que tras obtenerse el resultado de los métodos de scrapping se pasa a realizar la lista que seguidamente se pasa al método *comparar(),* tras haber sido ordenada por orden de precio.

```
public List <productos_tabla> getListMejorPrecio(String nombreProducto){

    List <productos_tabla> listTabla = new ArrayList<productos_tabla>();
    String [] links = proPers.getLinksyFoto(nombreProducto);

    scrapping_Chain(links[0]);
    scrapping_Retto(links[1]);
    scrapping_BikeInn(links[2]);
    scrapping_Fabregues(links[3]);
    scrapping_Mantel(links[4]);
    scrapping_Probike(links[5]);
    scrapping_Wiggle(links[6]);
    scrapping_BikeShop(links[7]);
    scrapping_Deporvillage(links[8]);

    System.out.println(links.toString());
    productos_tabla pTabla = new productos_tabla();
    pTabla.setNombreproducto(nombreProducto);
    //Chain Reaction
    if(precioChain!="")
        listTabla.add(new productos_tabla(nombreProducto,precioChain,"Chain","file:img/chain-reaction-cycles.png", links[0]));
    //Retto
    if(precioRetto!="")
        listTabla.add(new productos_tabla(nombreProducto,precioRetto,"Retto","file:img/logoRetto.png", links[1]));
    //BikeInn
    if(precioBikeInn!="")
        listTabla.add(new productos_tabla(nombreProducto,precioBikeInn,"BikeInn","file:img/logoBikeinn.jpg", links[2]));
    //Fabregues
    if(precioFabregues!="")
        listTabla.add(new productos_tabla(nombreProducto,precioFabregues,"Fabregues","file:img/logoFabregues.jpg", links[3]));
    //Mantel
    if(precioMantel!="")
        listTabla.add(new productos_tabla(nombreProducto,precioMantel,"Mantel","file:img/logoMantel.png", links[4]));
    //Probike
    if(precioProbike!="")
        listTabla.add(new productos_tabla(nombreProducto,precioProbike,"Probike","file:img/logoProbike.png", links[5]));
    //Wiggle
    if(precioWiggle!="")
        listTabla.add(new productos_tabla(nombreProducto,precioWiggle,"Wiggle","file:img/Wiggle_logo.png", links[6]));
    //BikeShop
    if(precioBikeShopp!="")
        listTabla.add(new productos_tabla(nombreProducto,precioBikeShopp,"BikeShop","file:img/logo.svg", links[7]));
    //Wiggle
    if(precioDeporVillage!="")
        listTabla.add(new productos_tabla(nombreProducto,precioDeporVillage,"Deporvillage","file:img/deporvillage_logo_2x.png", links[8]));

    System.out.println(listTabla.toString());

    listTabla.sort((productos_tabla p1, productos_tabla p2) -> p1.getPrecio().compareTo(p2.getPrecio()));//Ordenar tabla

    Productopersistente newPrecio = new Productopersistente();
    newPrecio.setNombrePersis(nombreProducto);
    newPrecio.setPrecioPersis(Float.parseFloat(listTabla.get(0).getPrecio()));
    newPrecio.guardarPrecio();

    System.out.println(listTabla.toString());
    return listTabla;
}
```

Figura 5.2 Obtención Lista de Precios

Una vez hemos obtenido este listado pasamos a leer la lista y se hace llamada al método *printDatos* con el que se muestran los datos dependiendo de la longitud de la lista, en el caso de que la lista tenga su longitud máxima se realizará el caso 9 del *switch:*

```
case 9:

    precio1.setText(listTabla.get(0).getPrecio()+"\u20ac");
    precio2.setVisible(true);
    precio2.setText(listTabla.get(1).getPrecio()+"\u20ac");
    precio3.setVisible(true);
    precio3.setText(listTabla.get(2).getPrecio()+"\u20ac");
    precio4.setVisible(true);
    precio4.setText(listTabla.get(3).getPrecio()+"\u20ac");
    precio5.setVisible(true);
    precio5.setText(listTabla.get(4).getPrecio()+"\u20ac");
    precio6.setVisible(true);
    precio6.setText(listTabla.get(5).getPrecio()+"\u20ac");
    precio7.setVisible(true);
    precio7.setText(listTabla.get(6).getPrecio()+"\u20ac");
    precio8.setVisible(true);
    precio8.setText(listTabla.get(7).getPrecio()+"\u20ac");
    precio9.setVisible(true);
    precio9.setText(listTabla.get(8).getPrecio()+"\u20ac");

    logo1.setImage(new Image(listTabla.get(0).getLogoTienda()));
    logo2.setVisible(true);
    logo2.setImage(new Image(listTabla.get(1).getLogoTienda()));
    logo3.setVisible(true);
    logo3.setImage(new Image(listTabla.get(2).getLogoTienda()));
    logo4.setVisible(true);
    logo4.setImage(new Image(listTabla.get(3).getLogoTienda()));
    logo5.setVisible(true);
    logo5.setImage(new Image(listTabla.get(4).getLogoTienda()));
    logo6.setVisible(true);
    logo6.setImage(new Image(listTabla.get(5).getLogoTienda()));
    logo7.setVisible(true);
    logo7.setImage(new Image(listTabla.get(6).getLogoTienda()));
    logo8.setVisible(true);
    logo8.setImage(new Image(listTabla.get(7).getLogoTienda()));
    logo9.setVisible(true);
    logo9.setImage(new Image(listTabla.get(8).getLogoTienda()));

    link1 = listTabla.get(0).getLink();
    link2 = listTabla.get(1).getLink();
    link3 = listTabla.get(2).getLink();
    link4 = listTabla.get(3).getLink();
    link5 = listTabla.get(4).getLink();
    link6 = listTabla.get(5).getLink();
    link7 = listTabla.get(6).getLink();
    link8 = listTabla.get(7).getLink();
    link9 = listTabla.get(8).getLink();

    bt2.setVisible(true);
    bt3.setVisible(true);
    bt4.setVisible(true);
    bt5.setVisible(true);
    bt6.setVisible(true);
    bt7.setVisible(true);
    bt8.setVisible(true);
    bt9.setVisible(true);

    break;
```

Figura 5.3 Lectura e impresión por UI del listado

Gráfica Precios Históricos

Al hacer la comparación también mostramos por pantalla una gráfica *lineChart* con la que mostramos los precios históricos de ese producto visualizados por el usuario para saber si el producto está subiendo o bajando de precio con el tiempo, y lo hemos implementado con el código que podemos ver en la figura 5.4.

```
try{
    ObservableList <XYChart.Data> precioHistorico=FXCollections.observableArrayList();

    List <Producto> productosHist = proPerst.getPreciosHistoricoProductos(modelo.getValue());

    for(Producto p : productosHist){
        precioHistorico.add(new XYChart.Data(p.getFecha(),p.getPrecio()));
    }

    XYChart.Series XYSeriesPrecioHist =new XYChart.Series();
    XYSeriesPrecioHist.setName("Mejores Precios historico");
    XYSeriesPrecioHist.setData(precioHistorico);

    histPrecioGen.setTitle("Precios Historicos");
    histPrecioGen.getData().setAll(XYSeriesPrecioHist);

    for(final XYChart.Series<String, Float> serie: histPrecioGen.getData()){
        for (final XYChart.Data <String, Float> datas : serie.getData()) {
                Tooltip toolTip=new Tooltip();
                toolTip.setText(String.valueOf(datas.getXValue() + "\n" +
                            "Precio: " + datas.getYValue()+"\u20ac"));
                Tooltip.install(datas.getNode(),toolTip);

        }
    }
}catch (NullPointerException e) {
    System.out.println("Aun no hay datos de precios.");
}
```

Figura 5.4 Realización de la gráfica de precios Históricos

Scrapping

A continuación, vamos a explicar cómo hemos realizado el análisis de las webs para conocer el campo necesario a extraer, para poder obtener el precio del producto buscado, y seguidamente explicaremos como implementar el algoritmo de scrapping

Además, con el algoritmo que vamos a realizar, extraeremos el precio del producto y el nombre y la descripción del producto las extraeremos del archivo de datos que realizaremos más adelante.

- **Análisis de webs.**

Para realizar el scrapping de las 7 webs que usaremos en nuestra aplicación vamos a analizar el código HTML que usan las diversas webs, para poder hacer el scrapping de cada una de las webs, que cada una de ellas como podremos ver usan unas etiquetas diferentes para cada uno de los datos que extraeremos

Seguidamente explicaremos para cada una de las webs que vamos a usar en la web cual es la etiqueta que vamos a usar, en estos casos buscaremos el producto “Manetas de cambio y freno Shimano Ultegra R8000”.

- **Chain Reaction**

En esta tienda se usa un diseño de listas constantemente y podemos ver en la figura 5.5. que en "li.crcPDPPrice" Chain Reaction incluye el PVP, el descuento asociado a la oferta y el precio con el descuento aplicado, este último es el que usaremos para el scrapping debido a que en esta web siempre se realizan descuentos, por lo que la etiqueta que usaremos en el scrapping es **"li.crcPDPPriceCurrent",** que se corresponde con el capo del precio actual. Pero al hacer el scrapping con esta etiqueta encontramos que el precio se encuentra en una etiqueta que no aparece al hacer f12, exactamente aparece en la clase **"crcPDPPriceHidden",** por lo que usaremos esta etiqueta para buscar el precio en el scrapping.

Figura 5.5 Analisis HTML Chain Reaction

- **Retto**

En el caso de Retto hemos podido apreciar que toda la organización es a base de y como podemos ver en la figura 5.6 en el "div.dtlDetPrecio" podemos encontrar en la etiqueta **"span.priceact"** el precio actual del producto en esta web. Por este motivo usaremos la etiqueta **"span.priceact"** para el scrapping.

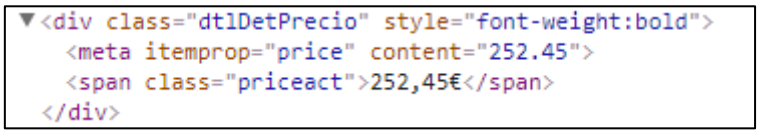

Figura 5.6 Análisis HTML Retto

- **Fabregues**

En el caso de esta tienda podemos decir que el precio de estas manetas se encuentra dentro de la etiqueta "div.price" como podemos ver en la figura 5.7 en el precio podremos encontrarlo con la etiqueta **"our_price_display" .**

Figura 5.7 Análisis HTML Fabregues

- **BikeInn**

BikeInn hace uso de la organización con *divs* y cómo podemos ver en la figura 5.8 el precio está dentro del **"datos_producto_precio"** y esta será la etiqueta que usaremos en el scrapping.

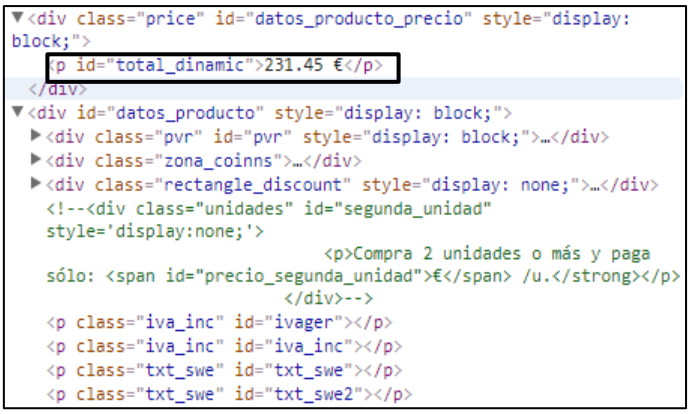

Figura 5.8 Análisis HTML Bikeinn

- **Probike**

En el caso de Probike hemos observado que sigue la línea del resto de webs, solo que en este caso el div que guarda la etiqueta final del precio es el "div.price-box" y la etiqueta específica para el precio es **"regular-price"** como podemos ver en la figura figura 5.9.

Figura 5.9 Análisis HTML Probike

- **Mantel**

Mantel utiliza en este caso "div#price-row" como contenedor para el precio y exactamente utiliza la etiqueta **"price-title"** para el precio que se muestra en pantalla, como podemos ver en la figura 5.10

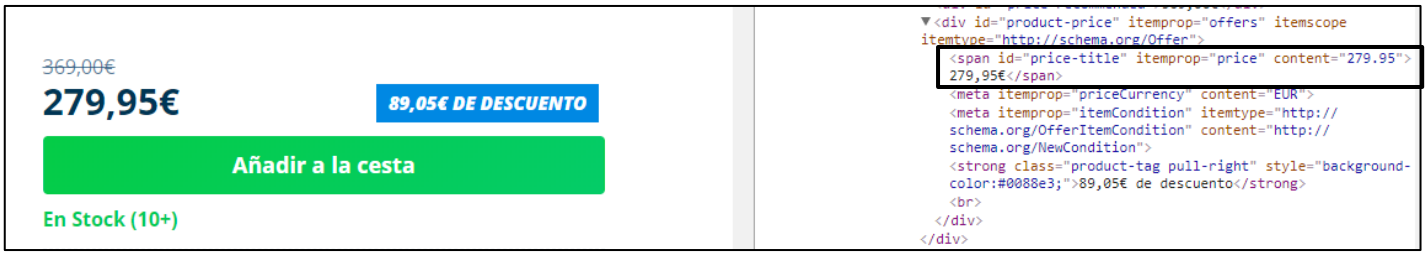

Figura 5.10 Análisis HTML Mantel

- **Wiggle**

En el caso de Wiggle hemos podido observar que que el div contenedor del precio es *div.bem-stickers* y que la etiqueta del precio que se muestra por pantalla es **"div.bem-pricing__product-price.js-unit-price"** como podemos ver en la figura 5.11.

```
▶<div class="bem-stickers">…</div>
 <span class="bem-product-price__expiry js-promo-ends hide">
 </span>
 <p class="bem-pricing__product-price js-unit-price" data-
 default-value="233,90 €">233,90 €</p> == $0
```

Figura 5.11 Análisis HTML Wiggle

- **BikeShop**

En el caso de la tienda BikeShop hemos podido comprobar que el precio actual del producto lo tienen almacenado dentro del id "producto-price", como se puede apreciar en la figura 4.12, por lo que usaremos esta etiqueta a la hora de realizar el scrapping.

```
<div itemprop="price" content="79.99">
  <div id="product-price" class="productPageContentInfosTopPrices_price" style="font-size: 38.4px;">
                  79,99 €            </div>
  <div id="product-price-instead" class="productPageContentInfosTopPrices_instead">
                   Precio de venta recomendado*: 119,90 €                </div>
```

Figura 5.12 Análisis HTML BikeShop

- **Deporvillage**

En cuanto a Deporvillage hemos visto que el precio actual de sus productos está alojado dentro del *div.price-box pero mas exactamente esta en el span.price*, por lo que a la hora de hacer el algoritmo haremos uso de la clase *Price*.

```
▼<div class="price-box">
  ▼<span class="regular-price" id="product-price-409713">
     <span class="price">239,95 €</span>
     <span class="m-l-xs t-price">IVA incl.</span>
   </span>
 </div>
```

Figura 5.13 Análisis HTML Deporvillage

- **Implementación de Scrapping**

Con el objetivo de extraer el precio de las webs hemos diseñado un método por cada web que vamos a usar, adicionalmente hemos realizado un método de conversión de tipos, debido a que hay webs que usan el formato decimal americano y otros que usan el método europeo.

Para la tienda Chain Reaction hemos utilizado el código de la figura 4.12. y podemos ver que con "`chain.getElementsByClass("crcPDPPriceHidden").first();`" extraemos el precio del campo "crcPDPPricehidden", y además los campos señalados en la figura 5.14. se repiten en todas las tiendas.

```
public static String scrapping_Chain(String url_chain){

    Document chain = getHtml(url_chain);
    //Obtención del precio
    Element precio = chain.getElementsByClass("crcPDPPriceHidden").first();
    precioChain = cambio_coma_Simbolo(precio.text());
    return precio.text();

}
```

Figura 5.14 Implementación Scrapping

El elemento precio es lo único que, en cada una de las tiendas varia, debido a que cada una tiene una etiqueta de búsqueda diferente, por lo que variara por las siguientes etiquetas en cada una de las tiendas:

- Retto: `chain.getElementsByClass("priceact").first();`
- BikeInn: `chain.getElementById("datos_producto_precio");`
- Fabregues: `chain.getElementById("our_price_display");`
- Mantel: `chain.getElementById("price-title");`
- Probike: `chain.getElementsByClass("regular-price").first();`
- Wiggle: `chain.getElementsByClass("bem-pricing__product-price js-unit-price").first();`
- BikeShop: `chain.getElementById("product-price");`
- Deporvillage: `chain.getElementsByClass("price").first();`

Filtrar Por tienda

Una vez el usuario a elegido el producto tiene la opción de filtrar por tiendas y de esta manera solo ver la comparación entre las tiendas deseadas. Una vez el usuario haga click sobre el botón *filtrar* se activara el método *filtrar()*.

En primer lugar, se comprobará que el usuario haya seleccionado correctamente el artículo, que se no se hayan seleccionado todas las tiendas para filtrar y que no se haya seleccionado ninguna tienda para filtrar, y lo hemos implementado como se puede observar en la figura 5.15.

```
String mensajeError ="";
if(marca.getValue()==null) mensajeError+="Debeseleccionar una marca de producto. \n";
if(tipo.getValue()==null) mensajeError+="Debe seleccionar un tipo de producto. \n";
if(modelo.getValue()==null) mensajeError+="Debe seleccionar un modelo de producto.";
if(!chainCheck.isSelected() && !rettoCheck.isSelected() &&!bikeinnCheck.isSelected()
        && !fabreguesCheck.isSelected() && !mantelCheck.isSelected() && !probikeCheck.isSelected() && !wiggleCheck.isSelected())
    mensajeError+="Ha seleccionado eliminar todos nuestras tiendas soportadas.";
if(chainCheck.isSelected() && rettoCheck.isSelected() &&bikeinnCheck.isSelected()
        && fabreguesCheck.isSelected() && mantelCheck.isSelected() && probikeCheck.isSelected() && wiggleCheck.isSelected())
    mensajeError+="Ha seleccionado visualizar todas las tiendas ya mostradas.";
if(mensajeError.length() >0){
    Alert alerta = new Alert(AlertType.ERROR);
    alerta.setTitle("Error en la selección");
    alerta.setContentText(mensajeError);
    alerta.showAndWait();
}
```

Figura 5.15 Comprobaciones de filtrado

En caso de que la variable *mensajeError* tras las comprobaciones siga vacío, seguirá la ejecución del método y entonces se obtendrá la lista de comparación y seguidamente eliminaremos de esta lista las tiendas que el usuario hay dejado sin seleccionar para filtrar, de esta forma dejaremos en la lista únicamente las tiendas seleccionadas por el usuario. Dicha comprobación y eliminación de la lista lo hemos realizado mediante el siguiente código de la figura 5.16

```
if(!chainCheck.isSelected()){
    for(productos_tabla pro : listFiltrado){
        if(pro.getTienda()=="Chain") prot=pro;
    }
    listFiltrado.remove(prot);
}
if(!rettoCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "Retto";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!bikeinnCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "BikeInn";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!fabreguesCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "Fabregues";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!mantelCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "Mantel";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!probikeCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "Probike";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!wiggleCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "Wiggle";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!bikeShopCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "BikeShop";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}
if(!deporVillageCheck.isSelected()){
    Predicate<productos_tabla> productoFilt = productos_tablas -> productos_tablas.getTienda() == "Deporvillage";
    listFiltrado.removeIf(productoFilt);
    System.out.println("Producto elim: "+listFiltrado.toString());
}

printDatos(listFiltrado);
```

Figura 5.16 Comprobaciones y eliminación de tiendas al filtrar

Tras obtener la lista filtrada pasamos a leer e imprimir por la interfaz de usuario los datos de la lista al igual que en la comparación anteriormente explicada, pero en este caso con la particularidad de que se desactivan los campos que no van a ser rellenados al haber filtrado, como se puede observar en la figura 5.17.

```
case 3:

    precio1.setText(listTabla.get(0).getPrecio()+"\u20ac");
    precio2.setVisible(true);
    precio2.setText(listTabla.get(1).getPrecio()+"\u20ac");
    precio3.setVisible(true);
    precio3.setText(listTabla.get(2).getPrecio()+"\u20ac");
    precio4.setVisible(false);
    precio5.setVisible(false);
    precio6.setVisible(false);
    precio7.setVisible(false);
    precio8.setVisible(false);
    precio9.setVisible(false);


    logo1.setImage(new Image(listTabla.get(0).getLogoTienda()));
    logo2.setVisible(true);
    logo2.setImage(new Image(listTabla.get(1).getLogoTienda()));
    logo3.setVisible(true);
    logo3.setImage(new Image(listTabla.get(2).getLogoTienda()));
    logo4.setVisible(false);
    logo5.setVisible(false);
    logo6.setVisible(false);
    logo7.setVisible(false);
    logo8.setVisible(false);
    logo9.setVisible(false);

    link1 = listTabla.get(0).getLink();
    link2 = listTabla.get(1).getLink();
    link3 = listTabla.get(2).getLink();

    bt2.setVisible(true);
    bt3.setVisible(true);
    bt4.setVisible(false);
    bt5.setVisible(false);
    bt6.setVisible(false);
    bt7.setVisible(false);
    bt8.setVisible(false);
    bt9.setVisible(false);

    break;
```

Figura 5.17 Caso 3 del Switch utilizado el filtrado

5.2.2. Capa de Persistencia

En este apartado os vamos a explicar cómo hemos realizado la capa de persistencia, para la cual hemos hecho uso de Hibernate y H2.

- **Diseño**

Con el objetivo de realizar la base de datos hemos realizado el diagrama de clases UML [19] , que representa las tablas que se usaran para la base de datos.

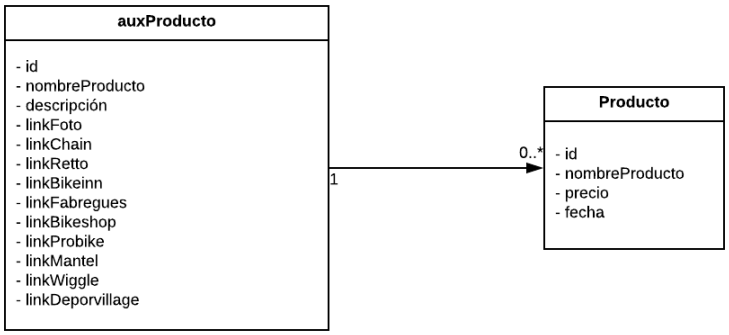

Figura 5.18 Diagrama de clases BD

- **Implementación**

Para integrar Hibernate+H2 necesitamos implementar el archivo *hibernate.cfg.xml* en el que hay que poner la configuración de Hibernate, como es el usuario, contraseña, driver de la conexión, URL de la DB, etc. Este archivo lo hemos configurado tal y como se puede observar en la figura 5.18.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:./src/bd/tfgDB</property>
        <property name="hibernate.connection.username">tfg</property>
        <property name="hibernate.connection.password">"tfg"</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>
    </session-factory>
</hibernate-configuration>
```

Figura 5.19 Archivo Hibernate.cfg.xml

Para la persistencia de los datos hemos creado el método *Productopersistente* donde hemos implementado los diversos métodos con los que guardar un producto, guardar los precios, obtener listados de marcas, obtener listados de categorías, obtener listados de tipos de productos, obtener la lista de mejores precios, etc.

Para ello en primer lugar hay que iniciar una conexión, figura 5.19, con la base de datos y una sesión como podemos ver en la figura 5.20.

```
private static SessionFactory inicioSesion(Class clase){
    return new Configuration().configure("/bd/hibernate.cfg.xml")
            .addAnnotatedClass(clase)
            .buildSessionFactory();
}
```

Figura 5.20 Inicio de conexión BD

```
//Inicio de Sesion en la BD
SessionFactory iniciarSesionBD = inicioSesion(Aux_producto.class);
Session sesion = iniciarSesionBD.openSession();
sesion.getTransaction().begin();
```

Figura 5.21 Inicio de sesión BD

Como podemos ver en la figura 4.18 cuando llamamos a *inicioSesion(...)* especificamos la clase que se usara como esquema para buscar en la BD, llamando a la tabla *Aux_producto* en el caso de la figura 4.18.

Una vez se ha iniciado la sesión, pasamos a hacer la transacción en la base de datos y la extracción de la información en el formato deseado para cada método:

- **Guardar Producto**: Como podemos observar en la figura 5.21, se crea el producto de la clase *Aux_producto* y seguidamente se guarda en la base de datos.

```
Aux_producto auxProducto =new Aux_producto();
auxProducto.setNombreProducto(nombrePersis);
auxProducto.setMarca(marcaPersis);
auxProducto.setCategoria(categoriapersis);
auxProducto.setTipo(tipoPersis);
auxProducto.setDescripción(descripcionPersis);
auxProducto.setLinkChain(linkChainPersis);
auxProducto.setLinkRetto(linkRettoPersis);
auxProducto.setLinkFabregues(linkFabreguesPersis);
auxProducto.setLinkBikeInn(linkBikeInnPersis);
auxProducto.setLinkProbike(linkProbikePersis);
auxProducto.setLinkMantel(linkMantelPersis);
auxProducto.setLinkWiggle(linkWigglePersis);
auxProducto.setFotoProducto(fotoProducto);

seision_aux.save(auxProducto);
System.out.println(auxProducto.toString());
seision_aux.getTransaction().commit();
```

Figura 5.22 Guardar producto

- **Guardar Precio**: Cuando guardamos el precio, también guardamos la fecha del día en la que realizamos la búsqueda, como se puede ver en la figura 5.22.

```
//Crear producto
Producto producto = new Producto();
producto.setNombreProducto(nombrePersis);
producto.setPrecio(precioPersis);
    //Guardamos la fecha del dia en que se busca el producto
LocalDate hoy = LocalDate.now();
String fecha = (hoy.getDayOfMonth()+"/"+hoy.getMonthValue()+"/"+hoy.getYear());
producto.setFecha(fecha);

sesion.save(producto);
System.out.println(producto.toString());
sesion.getTransaction().commit();
```

Figura 5.23 Guardar precio y fecha

- **Obtener listado de precios Histórico**: Al buscar los precios históricos para rellenar la gráfica de precios históricos hacemos un *select* a la base de datos como podemos ver en la figura 5.23 y seguidamente se crea un *list<String>*.

```
//Obtención de datos
List <Producto> precioFecha = new ArrayList();
List listProductos = sesion.createQuery( "from Producto where nombreProducto ='"+nombre_producto+"'" ).list();
precioFecha=(List <Producto>) listProductos;
```

Figura 5.24 Obtención precios históricos

- **Obtener Listado de marcas:** en el caso de querer el listado de marcas para el *choicebox* de marcas, como podemos ver en la figura 5.24, además también usamos un hashSet para eliminar los valores duplicados.

```
//Obtención de datos
List listProductos = sesion.createQuery( "select marca from Aux_producto where categoria ='"+categoria+"'" ).list();
System.out.println(listProductos.toString());
HashSet <String> sinDup = new HashSet<String>(listProductos);
List <String> list_Marcas = new ArrayList <String>();
list_Marcas.addAll(sinDup);
```

Figura 5.25 Obtención del listado de marcas

- **Obtener listado de categorías:** Cuando iniciamos la aplicación llamamos a este método para rellenar el choiceBox de categoría, con categorías como ropa o componente, y lo implementamos como se puede ver en la figura 5.25.

```
//Obtención de datos
List listProductos = sesion.createQuery( "select categoria from Aux_producto" ).list();
HashSet <String> sinDup = new HashSet<String>(listProductos);
List <String> list_categorias = new ArrayList <String>();
list_categorias.addAll(sinDup);
```

Figura 5.26 Obtención del listado de categorías

- **Obtener dirección de la imagen del producto:** Con el objetivo de visualizar por pantalla la foto del producto se hace un *select* a la base de datos y obtenemos un *String* con el link de la imagen como podemos ver en la figura 5.26.

```
//Obtención de datos
String fotoProducto="";
List listProductos = sesion.createQuery( "select fotoProducto from Aux_producto where nombreProducto ='"+nombreProducto+"'" ).list();
System.out.println(listProductos.toString());
List <String> list_links = (List <String>) listProductos; //El listado unicamente tendrá un valor
fotoProducto = list_links.get(0);
```

Figura 5.27 Obtención link de la foto del producto

- **Obtener un listado con los nombres de productos**: Con el objetivo de completar el choiceBox de nombre de producto habiendo seleccionado el usuario tanto la categoría, la marca y el tipo de producto, y ha sido implementada la petición de la siguiente manera:

```
//Obtención de datos
List listProductos = sesion.createQuery( "select nombreProducto from Aux_producto where marca ='"+marca+
        "' and categoria = '"+ categoria +"' and tipo = '"+modalidad+"'" ).list();
HashSet <String> sinDup = new HashSet<String>(listProductos);
List <String> list_Modalidades = new ArrayList <String>();
list_Modalidades.addAll(sinDup);
```

Figura 5.28 Listado de nombres de productos

- **Obtener la descripción del producto:** El usuario al comparar el producto entre las diversas webs acciona la petición de la descripción y se procede a la transacción como se puede ver en la figura 5.28.

```
//Obtención de datos
String descripcion="";
List listProductos = sesion.createQuery( "select descripción from Aux_producto where nombreProducto ='"+nombreProducto+"'" ).list();
System.out.println(listProductos.toString());
List <String> list_links = (List <String>) listProductos;
descripcion = list_links.get(0);
```

Figura 5.29 Obtención de la descripción

- **Obtener listado de tipos de productos:** Cuando el usuario decide una categoría y una marca tiene que elegir un tipo de producto y entonces realiza la consulta que podemos ver en la figura 5.29, extrayendo un listado de los tipos de productos que cumplan con los requisitos del usuario.

```
//Obtención de datos
List listProductos = sesion.createQuery( "select tipo from Aux_producto where marca ='"+marca
        +"' and categoria = '"+ categoria +"'" ).list();
HashSet <String> sinDup = new HashSet<String>(listProductos);
List <String> list_Modalidades = new ArrayList <String>();
list_Modalidades.addAll(sinDup);
```

Figura 5.30 Obtención de un listado de tipos de productos

Tras realizar las consultas a la base de datos hay que realizar la confirmación y cierre de la transacción junto al cierre de la sesión, como podemos ver en la figura 5.30

```
//Cierre de sesión BD
sesion.getTransaction().commit();
sesion.close();
iniciarSesionBD.close();
```

Figura 5.31 Commit, cierre de sesión y conexión BD

Por otro lado, hemos tenido que realizar la base de datos inicial para cargar los productos que el usuario podrá elegir, para ello hemos cargado un total de 45 productos, poniendo como ejemplo de introducción la figura 5.31.

```
Productopersistente pr;
pr=new Productopersistente();
pr.setNombrePersis("Bicicleta Carretera Orbea Orca M30 2018");
pr.setCategoriapersis("Bicicleta"); pr.setTipoPersis("Carretera");
pr.setMarcaPersis("Orbea");
pr.setDescripcionPersis("Esta bicicleta monta el cuadro Orbea Orca OME de carbono y el grupo Shimano 105 11v");
pr.setFotoProducto("https://www.sanferbike.com/tiendaonline/49442-thickbox_default/orbea-orca-m30-2018.jpg");
pr.setLinkProbikePersis("https://www.probike.com/bicicleta-carretera-orbea-orca-m30-2018.html");
pr.guardarProducto();
```

Figura 5.32 Inyección de datos en la BD

Como se puede apreciar en la figura 5.31, para ese producto no se introduce el link de todas las webs soportadas, debido a que las webs no incluidas, no tienen en su web disponible dicho producto.

5.2.3. Capa de Presentación

A continuación, vamos a explicar cómo hemos realizado la Interfaz gráfica empleada en esta aplicación.

Para la implementación hemos hecho uso de JavaFX y del programa SceneBuilder [20] para el diseño de la aplicación. Al usar JavaFx la interfaz está implementada sobre un archivo .xml, en nuestro caso es *ciclistas.xml* y el diseño está hecho en un archivo .css, que en nuestro caso dejaremos el archivo .css que viene por defecto con JavaFX.

Tras realizar el diseño con SceneBuilder la ventana de la Aplicación a quedado de la siguiente forma:

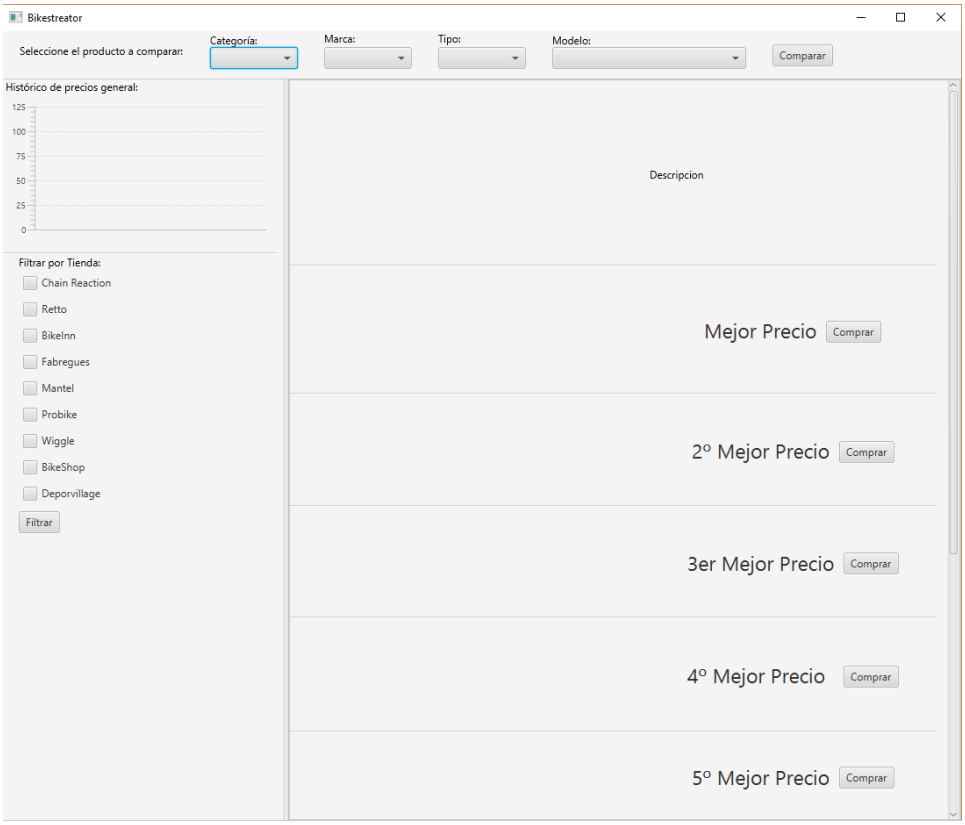

Figura 5.33 Interfaz de usuario implementada

Alertas

Con el objetivo de mejorar la experiencia del usuario hemos implementado unas alertas que saltarán cuando el usuario quiera comparar o filtrar un producto y no haya seleccionado algo correctamente, y de esta forma prevenir que tanto no se produzcan posibles excepciones y por lo tanto fallos en la ejecución de la aplicación y que el usuario sepa en todo momento si se está ejecutando correctamente o no el programa.

En los casos en los que se cumplan los siguientes casos se ejecutara una excepción:

- No selección de un valor en los desplegables de selección del producto:

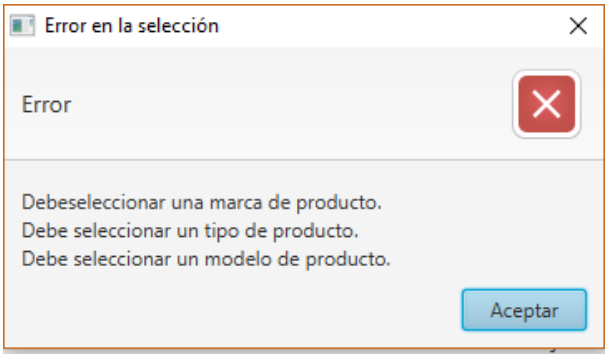

Figura 5.34 Alerta por no selección de producto

- Filtrar sin elegir tiendas:

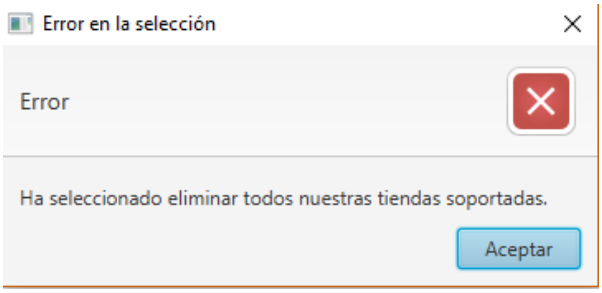

Figura 5.35 Alerta al dejar sin seleccionar todas las tiendas y querer filtrar

- Filtrar habiendo seleccionado todas las tiendas:

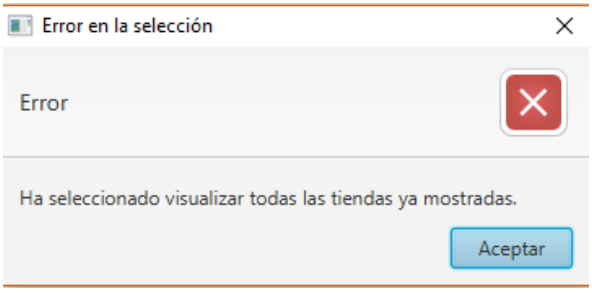

Figura 5.36 Alerta al querer filtrar queriendo visualizar todas las tiendas.

6. Ejemplos de Uso

A continuación, vamos a mostrar un ejemplo de uso basándonos en el escenario 1 en el que el usuario quiere comparar el neumático “Continental Grand Prix SII”, pero con unas pequeñas modificaciones debido a que vamos a mostrar también posibles situaciones en las que el usuario realice algún error, y de esta forma mostrar como la aplicación nos informa de los posibles errores.

En primer lugar, el usuario seleccionará primero una categoría, una marca, un tipo de producto y por último elegirá el modelo deseado.

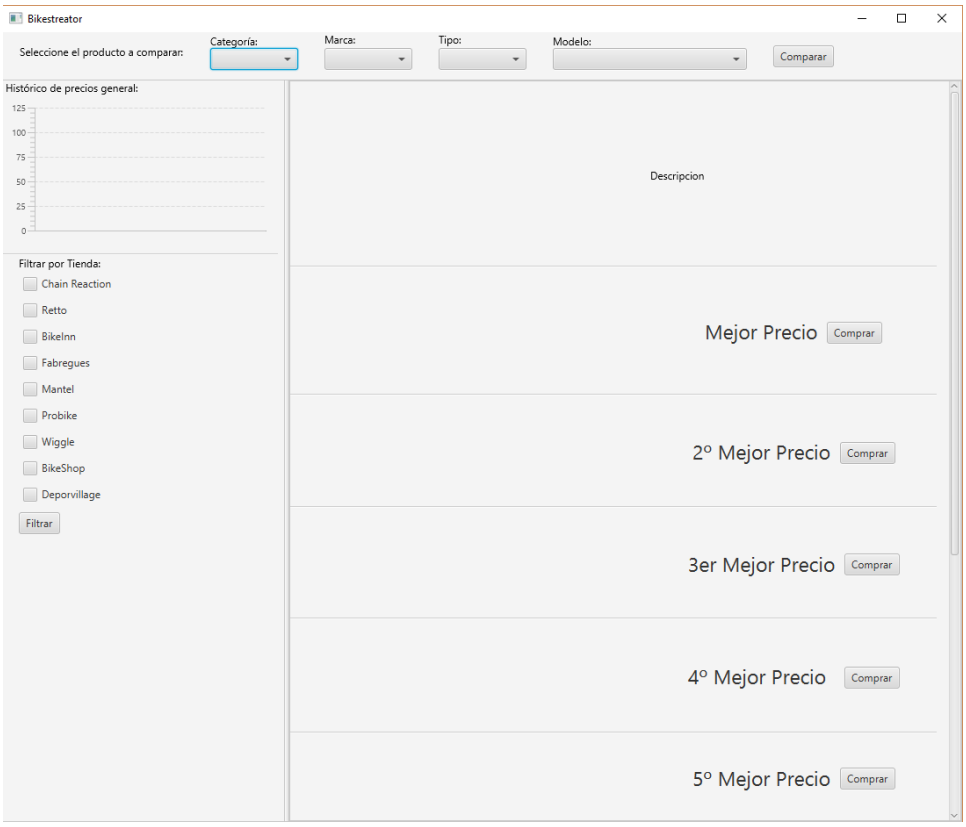

Figura 6.1 Selección del producto

Tras haber seleccionado el producto el usuario hace click sobre el botón *Comparar* y seguidamente aparecen por pantalla el producto y los diferentes precios relacionados con su respectiva tienda y un botón de compra, además también se muestra una gráfica donde se puede ver la tendencia del precio a lo largo del tiempo, como se puede observar en la figura 6.2

El usuario no quiere comprar en la tienda Retto, por lo que tendrá que seleccionar todas las tiendas excepto Retto, pero al seleccionarlas todas, por error ha seleccionado Retto también y al hacer click sobre el botón *Filtrar* Salta una alerta avisando de que se han seleccionado todas las tiendas y que por lo tanto no habría cambios en la muestra de datos, como se puede ver en la figura 6.3.

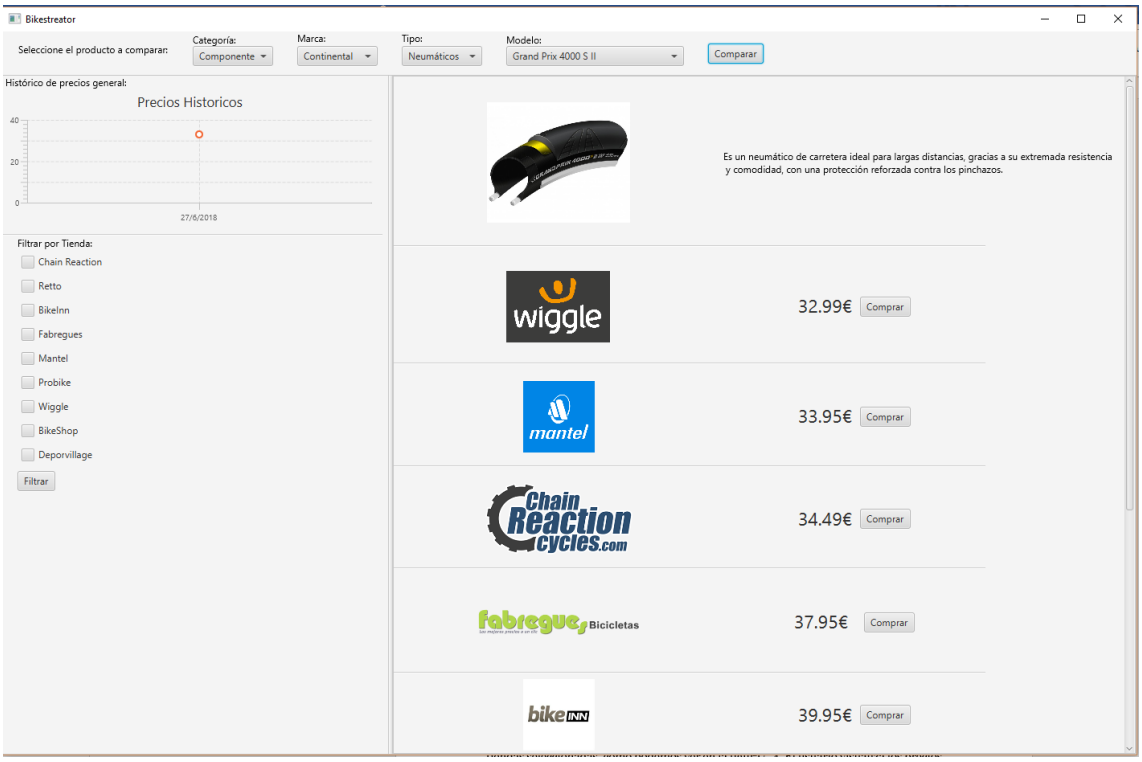

Figura 6.2 Comparación de productos

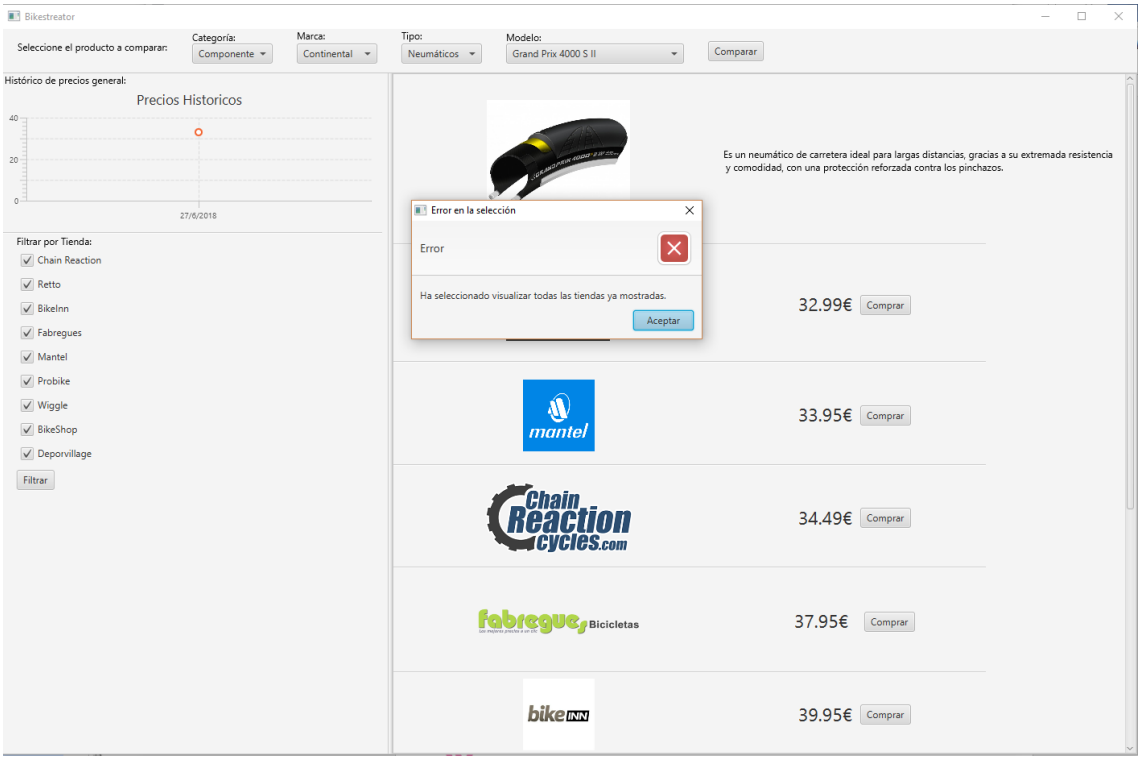

Figura 6.3 Alerta por seleccionar todas las tiendas

El usuario al ver este error se da cuenta de que ha seleccionado también la tienda Retto y, por lo tanto, hace click sobre el botón *Aceptar* y seguidamente deselecciona la Tienda Retto y vuelve a hacer click sobre el botón *Filtrar* y le aparecen por pantalla las tiendas seleccionadas, como podemos ver en la figura 6.4. El usuario visualiza los precios y escoge la tienda que más le gusta y hace click sobre el botón comprar y es redirigido a la tienda seleccionada para que proceda a la compra del producto.

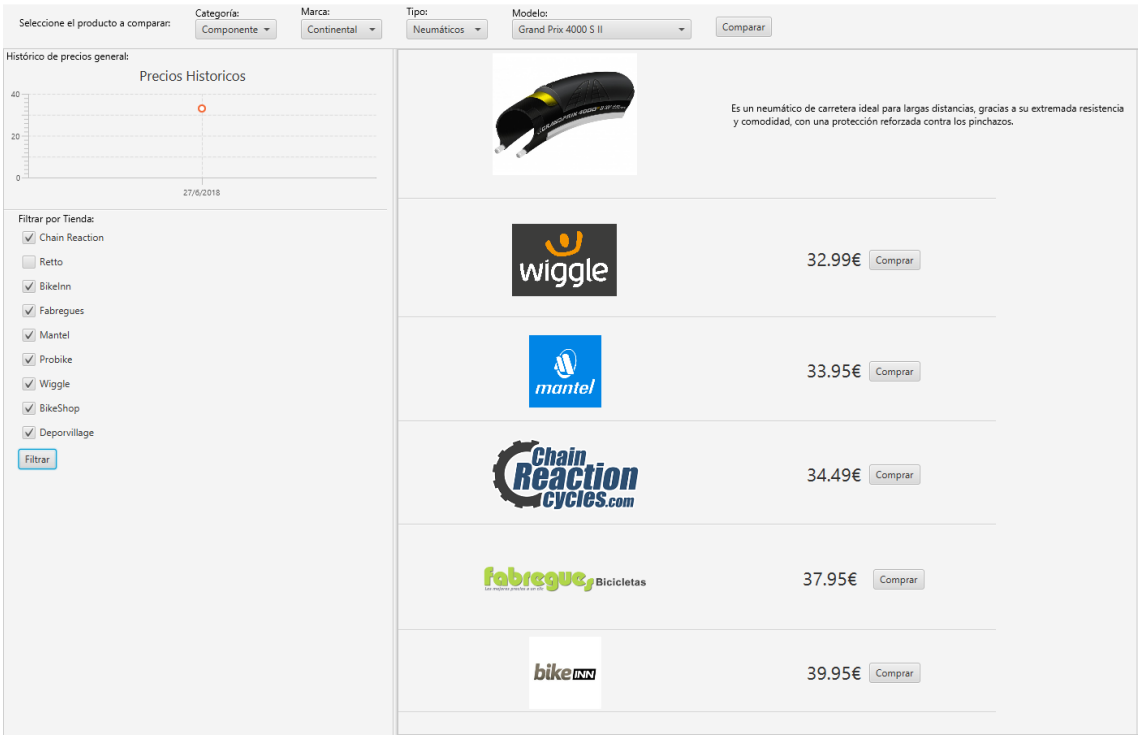

Figura 6.4 Muestra de filtrado

7. Conclusión

En este apartado vamos a explicar cuáles son las conclusiones a las que hemos llegado tras la realización de este trabajo de fin de grado, así como las mejoras que podríamos aplicar sobre el en un futuro.

7.1. Conclusiones

A lo largo de este trabajo hemos comentado que el usuario tiene la necesidad de tener una aplicación que poniendo el producto que el usuario busque la aplicación compare los precios de los productos y seguidamente los muestre por pantalla ordenados.

En un inicio la aplicación únicamente íbamos a implementar el scrapping sobre 7 tiendas, pero en la finalización del trabajo fueron añadidas 2 tiendas debido a que en la realización de la base de datos fueron encontradas 2 tiendas muy importantes ciclistas que no habían sido incluidas en el alcance. De esta forma la aplicación haciendo uso de los algoritmos de scrapping, anteriormente descritos, y la base de datos realizada, hemos podido realizar con total éxito la aplicación.

Por lo anteriormente expuesto podemos decir que la aplicación realizada ha conseguido cumplir con todos los requisitos expuestos a lo largo de esta memoria, y podría ser utilizada por multitud de ciclistas con total éxito.

7.2. Mejoras

A continuación, vamos a explicar posibles mejoras que realizar a la aplicación de cara al futuro:

- Hacer que la búsqueda del producto sea simplemente escribiendo el usuario el nombre del producto que busca y que la aplicación lo busque de forma automática.
- Realizar una hoja de estilos mucho más atractiva para el usuario, mejorando la interfaz gráfica.
- Hacer que la aplicación sea compatible con más plataformas como: web, iOS, Android y MacOS.
- Implementar una base de datos online y usar esta base de datos para todos los usuarios, en vez de tener la base de datos local.

8. Bibliografía

[1] Isabel Fernandez, 12 Diciembre 2017. [En línea]. Available: https://www.nobbot.com/redes/internet-es-tu-aliado-en-rebajas-mejores-comparadores-de-precios/..

[2] «Kayak,» [En línea]. Available: https://www.kayak.es/about. [Último acceso: 15 Mayo 2018].

[3] «Skyscanner,» [En línea]. Available: https://www.skyscanner.es/prensa/aboutskyscanner. [Último acceso: 15 Mayo 2018].

[4] «Rastreator,» [En línea]. Available: http://www.rastreator.com/quienes-somos.aspx. [Último acceso: 15 Mayo 2018].

[5] «Google Shopping,» [En línea]. Available: https://www.google.es/shopping?hl=es_ES. [Último acceso: 15 Mayo 2018].

[6] «Corebicycle,» [En línea]. Available: https://www.corebicycle.com/. [Último acceso: 15 Mayo 2018].

[7] «ComparaCiclismo,» [En línea]. Available: http://www.comparaciclismo.com. [Último acceso: 15 Mayo 2018].

[8] «NBICI,» [En línea]. Available: http://www.nbici.com/. [Último acceso: 15 Mayo 2018].

[9] «Java,» [En línea]. Available: https://www.java.com/es/. [Último acceso: 27 Febrero 2018].

[10] «Eclipse,» [En línea]. Available: https://www.eclipse.org/. [Último acceso: 15 Mayo 2018].

[11] «JavaFX,» [En línea]. Available: http://www.oracle.com/technetwork/es/java/javafx/downloads/index.html. [Último acceso: 27 Febrero 2018].

[12] «Hibernate,» [En línea]. Available: http://hibernate.org/. [Último acceso: 27 Febrero 2018].

[13] «H2 Database,» [En línea]. Available: http://www.h2database.com/html/main.html. [Último acceso: 27 Febrero 2018].

[14] «JSoup,» [En línea]. Available: https://jsoup.org/. [Último acceso: 27 Febrero 2018].

[15] «No Solo Usabilidad,» [En línea]. Available: http://www.nosolousabilidad.com/manual/3.htm. [Último acceso: 20 Mayo 2018].

[16] «Google Forms,» [En línea]. Available: https://www.google.es/intl/es/forms/about/. [Último acceso: 15 Mayo 2018].

[17] «Leyes de la percepcion - UV,» [En línea]. Available: https://www.uv.es/asamar4/exelearning/21_las_leyes_de_la_gestalt.html. [Último acceso: 27 Junio 2018].

[18] «Marco de Desarrollo de la Junta de Andalucia,» [En línea]. Available: http://www.juntadeandalucia.es/servicios/madeja/sites/default/files/historico/1.3.0/contenido-subsistemas-desarrollo-construccion-por-capas-aplicaciones-java.html. [Último acceso: 20 Junio 2018].

[19] A. Canchala, «UML, ejemplo sencillo sobre Modelado de un Proyecto,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/bb972214.aspx. [Último acceso: 15 Junio 2018].

[20] «SceneBuilder JavaFX,» [En línea]. Available: http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html. [Último acceso: 20 Febrero 2018].