

Universidad ORT Uruguay

Facultad de Ingeniería

# **Ingeniería de Software en la Práctica Obligatorio 1**

Agustín Hernandorena (233361)

Joaquín Lamela (233375)

Santiago Castro (168347)

<https://github.com/ORTISP/AhorroUY>

Entregado como requisito de la materia Ingeniería de  
Software en la Práctica

16 de junio de 2021

# Declaraciones de autoría

Nosotros, Joaquín Lamela, Agustín Hernandorená y Santiago Castro, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos Ingeniería de Software en la práctica;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.

## **Resumen**

En la actualidad, en el mundo a nivel general se están produciendo grandes cambios en diferentes ámbitos. Uno de ellos el cual es fundamental para la vida de la humanidad, es la economía. Si bien el esfuerzo impulsado por los diferentes países ha ayudado a solventar los problemas que se están viviendo por el desarrollo de la pandemia, de todas formas, la economía familiar se está viendo afectada. Este impacto se ve reflejado en: la falta de empleo, reducción en horas de trabajo, donde esta situación se ve agravada en familias numerosas. El entorno uruguayo no es ajeno a esto, y particularmente en nuestro país se está viviendo un tiempo de cambio, en donde el uruguayo promedio está tomando conciencia acerca del ahorro y busca los mejores lugares para realizar las compras de forma de ayudar a su bolsillo.

Algunos de los resultados obtenidos a través del análisis a informes científicos, información certera y entrevistas a calificados del tema, revelan que en Uruguay, se están tratando de realizar campañas para concientizar tanto a usuarios como empresas de lo que está sucediendo con el ámbito del ahorro. Estas campañas repercuten a la hora de realizar la especificación del sistema, de forma que se abren nuevos campos a desarrollar.

De esta manera, se busca mediante la investigación y el desarrollo de un aplicación *mobile*, promover prácticas de ahorro en las compras a supermercados y mini-supermercados, permitiendo al usuario comparar los diferentes precios para un determinado grupo de productos en los diferentes supermercados, así como saber la distancia a la que se encuentra cada uno de ellos teniendo en cuenta la localización del usuario.

# Índice general

<b>1. Análisis del problema</b>	<b>5</b>
1.1. Contexto . . . . .	5
1.1.1. ¿Quiénes son los potenciales usuarios? . . . . .	6
1.1.2. ¿Qué sabemos acerca de los usuarios? . . . . .	7
1.1.3. ¿En qué momento del día el usuario usará con mayor frecuencia la aplicación? . . . . .	7
1.1.4. ¿De qué forma el usuario tendrá el dispositivo al momento de hacer uso de la aplicación? . . . . .	8
1.1.5. Roles en la aplicación . . . . .	8
1.2. Alcance . . . . .	8
1.3. Especificación de historias de usuario . . . . .	9
1.3.1. Visualizar pantalla de inicio . . . . .	9
1.3.2. Registro . . . . .	10
1.3.3. Alerta de promociones . . . . .	11
1.3.4. Login . . . . .	13
1.3.5. Logout . . . . .	14
1.3.6. Búsqueda mediante la barra de navegación . . . . .	14
1.3.7. Búsqueda mediante código de barras . . . . .	16
1.3.8. Ordenar búsqueda por criterios . . . . .	17
1.3.9. Filtrado de productos según categoría . . . . .	19
1.3.10. Productos favoritos . . . . .	20
1.3.11. Cupones de descuento en compras . . . . .	22
1.3.12. Visualizar carrito de compras . . . . .	24
1.3.13. Obtener la mejor opción de compra para un conjunto de productos . . . . .	25
1.4. Dispositivos del celular utilizados . . . . .	27
<b>2. Gestión de proyecto</b>	<b>29</b>
2.1. <i>Metodología</i> . . . . .	29
2.2. <i>Scrum</i> . . . . .	29
2.2.1. Ceremonias . . . . .	30
2.2.2. Roles . . . . .	31
2.2.3. <i>Product Backlog</i> . . . . .	31
2.2.4. Definition of Done . . . . .	32
2.3. <i>Release plan</i> . . . . .	32
2.3.1. Condiciones de satisfacción . . . . .	32

2.3.2. Estimación de historias de usuario . . . . .	33
2.3.3. Largo de la iteración . . . . .	33
2.3.4. Estimación de velocidad . . . . .	34
2.3.5. Priorización de historias de usuario . . . . .	35
2.4. <i>Sprint planning</i> . . . . .	35
2.4.1. Sprint 1 . . . . .	36
2.5. Ejecución del Sprint 1 . . . . .	37
2.5.1. Sprint Review . . . . .	37
2.5.2. Sprint Retrospective . . . . .	38
2.5.3. Sprint 2 (Original) . . . . .	39
2.5.4. Sprint 2 (Revisado) . . . . .	40
2.5.5. Sprint review . . . . .	40
2.5.6. <i>Sprint retrospective</i> . . . . .	42
2.5.7. Sprint 3 (Original) . . . . .	44
2.5.8. Sprint 3 (Revisado) . . . . .	45
2.5.9. Sprint review . . . . .	46
2.5.10. <i>Sprint retrospective</i> . . . . .	48
2.5.11. Sprint 4 (Original) . . . . .	50
2.5.12. Sprint 4 (Revisado) . . . . .	51
2.5.13. Sprint 5 (Original) . . . . .	53
2.5.14. Sprint 5 (Revisado) . . . . .	54
2.5.15. Sprint 6 (Original) . . . . .	57
2.5.16. Sprint 6 (Revisado) . . . . .	58
2.6. Gestión de riesgos . . . . .	62
2.6.1. Minimización de riesgos al usar Scrum . . . . .	66
2.6.2. Análisis de probabilidad de ocurrencia . . . . .	67
<b>3. Gestión de la configuración</b>	<b>68</b>
3.1. Control de cambios y de versiones . . . . .	68
3.1.1. Estructura del repositorio y liberaciones identificadas . . . . .	68
3.2. Proceso de gestión de solicitud de cambios . . . . .	74
3.3. Herramientas de registro de defectos . . . . .	74
3.3.1. <i>SonarQube</i> . . . . .	75
3.3.2. <i>Designite</i> . . . . .	76
3.4. Pruebas de <i>software</i> . . . . .	86
3.4.1. Cobertura de pruebas . . . . .	86
3.4.2. <i>BusinessLogic</i> . . . . .	87
3.4.3. <i>BusinessLogicException</i> . . . . .	88
3.4.4. <i>BusinessLogicTest</i> . . . . .	88
3.4.5. <i>DataAccess</i> . . . . .	89
3.4.6. <i>DataAccessTest</i> . . . . .	90
3.4.7. <i>Domain</i> . . . . .	90
3.4.8. <i>DomainException</i> . . . . .	91
3.4.9. <i>BusinessLogicForPushNotification</i> . . . . .	91
3.4.10. <i>EntitiesForPushNotification</i> . . . . .	91
3.4.11. <i>RepositoryException</i> . . . . .	91

3.4.12. <i>WebApi</i> . . . . .	92
3.4.13. <i>WebApiTest</i> . . . . .	92
<b>4. Decisiones de Diseño</b>	<b>94</b>
4.1. Diseño de la arquitectura de la aplicación móvil . . . . .	94
4.1.1. Descripción General - MVVM Architecture . . . . .	94
4.1.2. Comunicación entre <i>View</i> y <i>ViewModel</i> . . . . .	96
4.1.3. Comunicación entre <i>ViewModel</i> y <i>Model</i> . . . . .	97
4.1.4. Separación de problemas . . . . .	99
4.1.5. Single Activity Architecture . . . . .	99
4.1.6. Navegación . . . . .	101
4.1.7. Librerías utilizadas . . . . .	102
4.2. UX . . . . .	105
4.2.1. <i>Navigation Drawer</i> . . . . .	105
4.2.2. Mensajes de error . . . . .	106
4.2.3. Carrito y productos favoritos . . . . .	106
4.2.4. <i>Home page</i> . . . . .	107
4.2.5. Sección mapa con supermercados . . . . .	108
4.3. Diseño de la arquitectura del Backend . . . . .	109
4.3.1. Diagrama de descomposición de los <i>namespaces</i> . . . . .	109
4.3.2. Diagrama general de paquetes ( <i>namespaces</i> ) . . . . .	112
4.3.3. Responsabilidades de cada paquete . . . . .	113
4.3.4. Mecanismo de acceso de datos . . . . .	115
4.3.5. Estructura de las tablas en la base de datos . . . . .	118
4.3.6. Justificación del diseño para cada paquete . . . . .	119
<b>5. Especificación API</b>	<b>126</b>
5.1. API REST . . . . .	126
5.1.1. Principios REST . . . . .	126
5.2. Descripción general de códigos de error . . . . .	128
5.3. Descripción de los resources de la API . . . . .	131
5.3.1. BestOptionController . . . . .	131
5.3.2. CategoryController . . . . .	132
5.3.3. ProductController . . . . .	132
5.3.4. PurchaseController . . . . .	134
5.3.5. SessionController . . . . .	135
5.3.6. TokenController . . . . .	136
5.3.7. UserController . . . . .	136
5.3.8. FavoritesController . . . . .	138
<b>6. Clean Code</b>	<b>140</b>
6.1. Nombres nemotécnicos . . . . .	140
6.2. Nombre de clases y métodos . . . . .	141
6.3. Funciones . . . . .	142
6.4. Comentarios . . . . .	143
6.5. Formateo . . . . .	143
6.5.1. Horizontal . . . . .	143

6.5.2. Vertical . . . . .	144
6.6. Pruebas unitarias . . . . .	145
<b>7. Instructivo de instalación</b>	<b>147</b>
7.1. Deployment Backend . . . . .	147
7.2. Aplicación Android . . . . .	151
Bibliografía . . . . .	152

# 1. Análisis del problema

## 1.1. Contexto

La compra de alimentos y artículos de aseo y limpieza básicos, es uno de los gastos más significativos de las finanzas familiares, es por eso, que es importante aprender a comprar de forma más inteligente para no salirnos del presupuesto.

La comida es una de las necesidades básicas del ser humano, por eso cuando pensamos en reducir el presupuesto en alimentación la primera reacción suele ser negativa, como si reducir el presupuesto fuera igual a bajar la calidad de nuestra alimentación, lo que implicaría una mayor predisposición para las enfermedades.

Pero no es así, se puede comer bien, y alimentarse bien, reduciendo el presupuesto, al punto de que una aplicación donde se pueda agregar el carrito que se realiza semanalmente, y permita generar un ahorro, sería fundamental para las finanzas familiares. Realmente, las diferencias de precios entre unos establecimientos y otros pueden ser muy importantes. También la ciudad donde se vive, influye en los mismos. Según la Organización de Consumidores y Usuarios (CUA), eligiendo las cadenas más baratas se pueden ahorrar hasta 52000 pesos uruguayos al año.

Cuando vamos a un supermercado físico a realizar nuestras compras sin tener un conocimiento previo de las promociones que se encuentran vigentes de los productos, se nos va a dificultar en gran medida encontrar esas promociones por diversos motivos: mucha gente en el supermercado, desconocimiento de la estructura del local edilicio, falta de tiempo, entre otros aspectos. Quizás, si estas promociones el usuario las recibe de antemano en su celular (o por cualquier otro mecanismo), se le facilitaría la obtención de ofertas interesantes en productos (por ejemplo: ofertas 3x2, descuentos que se generan mediante la generación de cupones), debido a que ya no estarían presentes los inconvenientes que se mencionaron arriba.

Dado el avance de las tecnologías, han surgido las compras *online* a supermercados ahorrando tiempo (al repetir la misma lista que ya se ha realizado), pero aumentando costos debido a envíos o por solo hecho de estar en diferentes aplicaciones móviles las cuales inducen un costo para los supermercados. También la compra *online*, permite reflexionar desde la tranquilidad del hogar de las diferentes necesidades, viendo qué necesitamos y qué es lo que ya tenemos en casa.

Finalmente, un punto importante que debemos remarcar en este momento es la crisis socioeconómica actual, producto del COVID-19, que ha generado cambios rápidos y disruptivos en las tendencias de consumo a nivel global. Las personas, por cuestiones sanitarias no pueden permanecer un lapso de tiempo muy prolongado en ambientes cerrados concurridos (como puede ser un supermercado en hora pico), el desempleo ha aumentado mucho, implicando un fuerte descenso en los ingresos familiares, teniendo como consecuencia el hecho de que las familias, con el fin de sostener su economía, tengan que buscar el mejor precio para una canasta básica de productos, sin necesariamente disminuir la calidad de los mismos, o desplazarse una gran distancia para encontrar los mejores precios.

Con respecto al último punto mencionado, se puede hacer énfasis en la diferencia de precios observada por una nota que publica el Gobierno cada dos semanas. En esta nota que se publica, se muestran los precios de alimentos de una canasta básica elaborada por el Sistema de Información de Precios al Consumidor (SIPC) y el Mercado Agrícola (MAM), con mínimos, máximos y promedios. Esta información, con valores actualizados al 24 de marzo de 2021, es difundida para ofrecer a la ciudadanía datos de interés para realizar las compras. Dicha tabla comparativa se presenta a continuación.

Producto	Unidad	Mín	Máx	Promedio	Devoto	Disco	El Clon	El Dorado	Macro	Ta - Ta	Tienda Inglesa
Aceite de girasol	900 cc	59	125	99	104	104	103	97	91	99	104
Aguja vacuna	1 Kg	220	421	257	230	230	—	—	265	278	272
Arroz blanco	1 Kg	22	65	50	52	52	38	52	42	49	54
Arvejas en conserva	300 grs	17	43	28	29	28	—	24	31	33	—
Azúcar blanco	1 Kg	36	58	41	41	41	39	40	41	40	41
Carne picada vacuna	1 Kg	199	419	283	279	279	—	—	314	270	309
Cocoa	500 grs	98	206	150	159	157	159	110	155	—	166
Dulce de leche	1 Kg	102	217	164	166	166	158	155	165	168	162
Fideos secos al huevo	500 grs	37	77	59	60	60	50	56	58	60	68
Galletitas al agua	140 grs	33	58	47	48	48	—	47	—	45	44
Harina trigo común 0000	1 kg	39	65	53	55	53	48	53	50	53	54
Huevos colorados	1/2 docena	50	75	63	63	63	—	60	61	67	69
Manteca	200 grs	54	86	71	69	69	74	65	74	68	66
Pan flauta	215 grs	35	60	54	—	55	—	—	—	—	—
Papel higiénico hoja simple	4 rollos 30 mts	45	119	82	91	91	88	73	94	83	91
Pollo entero fresco con menudos	1 kg	99	169	150	139	139	—	—	142	167	151
Pulpa de tomate	1 Lt	48	92	70	70	70	55	69	64	74	72
Sal fina yodada fluorada	500 grs	29	59	46	48	48	45	43	51	42	49
Yerba mate común	1 Kg	125	271	177	175	173	161	172	174	178	179

Lo que hace notar como hay gran diferencia de precios entre los diversos supermercados mencionados en la tabla, produciéndose así que la ida a un supermercado puede significar un gasto mayor con respecto a otro.

Es por todas estas razones, que es necesaria la creación de una aplicación que permita comparar los diferentes precios de un producto para diferentes supermercados. Pudiendo así, armar un determinado carrito, y utilizando la localización del individuo, brindarle la mejor opción para realizar la compra.

### 1.1.1. ¿Quiénes son los potenciales usuarios?

Los potenciales usuarios del sistema son personas residentes en la ciudad de Montevideo, Uruguay, que deseen realizar compras en supermercados consiguiendo el mejor precio para un determinado *set* de productos.

### **1.1.2. ¿Qué sabemos acerca de los usuarios?**

De los futuros usuarios, sabemos que residen en la ciudad de Montevideo, y que tal condición determina que la lengua nativa de los mismos sea el español.

Otro aspecto a destacar, es que *a priori* no podemos determinar una franja etaria de los potenciales usuarios, y esa condición determina que la aplicación debe ser lo suficientemente sencilla, debido a que personas de edad avanzada pueden llegar a hacer uso de la misma.

### **1.1.3. ¿En qué momento del día el usuario usará con mayor frecuencia la aplicación?**

Dado que la aplicación permite comparar los precios de determinados productos en supermercados de la ciudad de Montevideo, consideramos que un buen punto para dar respuesta a la pregunta planteada, es realizar un análisis exhaustivo de los diferentes supermercados existentes en dicha ciudad, detallando cuales son los horarios más concurridos durante los días de semana en cada uno de ellos.

Para obtener una visión mas amplia y objetiva de los horarios pico, consideramos tanto grandes superficies como es el caso de: Tienda Ingresa, Ta-Ta, Devoto, Disco y Macromercado, así como de supermercados mas pequeños como es el caso de: Martín Pescador, Kinko, Frigo, Frog y Planeta.

Por medio de las estadísticas de *Google* que se reflejan en la tabla 1.1, se puede observar algunos datos contundentes. Podemos ver como hay una tendencia marcada a realizar compras los días entre semana (lunes a viernes), preferentemente al mediodía en el periodo comprendido entre las 11 y 13 horas, así como también en el horario de la tarde-noche que comprende un rango de horas entre las 18 y 20. Sin embargo, ocurre que en los días sábados y domingos hay una leve variación con respecto a los días ya mencionados, notando una preferencia en la concurrencia al medio día en el rango de las 12 y 13 horas.

Supermercado	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
Disco	19 hs	12 hs	18 hs	13 hs	12 hs	12-13 hs	12-13 hs
Tienda Ingresa	18 hs	11 hs	11 hs	10-11-12 hs	18 hs	11-12 hs	19 hs
Martín Pescador	19 hs	20 hs	19 hs	19 hs	19-20 hs	19-20 hs	19 hs
Devoto Fresh Market	12 hs	11 hs	19 hs	12 hs	19 hs	12-13 hs	12 hs
Supermercado Frigo Sayago	19 hs	19 hs	19 hs	18-19 hs	19 hs	12 hs	12 hs
Supermercado Frog 4	12 hs	13 hs	11 hs	11 hs	17 hs	12 hs	13 hs
Macromercado Mayorista S.A	11-12 hs	11-12 hs	12 hs	11-12 hs	11-12 hs	12 hs	12 hs
Supermercado Ta-Ta	18 hs	18 hs	18 hs	18 hs	18 hs	12-13 hs	19 hs
Supermercado Kinko	11 hs	11 hs	13 hs	11 hs	19 hs	12 hs	19 hs
Supermercado Planeta	19 hs	12-13 hs	19 hs	12-13 hs	12 hs	13 hs	12-13 hs

Tabla 1.1: Horarios de mayor concurrencia en principales supermercados de la ciudad de Montevideo.

#### **1.1.4. ¿De qué forma el usuario tendrá el dispositivo al momento de hacer uso de la aplicación?**

Naturalmente un usuario de un teléfono móvil lo hace de forma vertical. Al no tratarse de un aplicación de vídeo ni de fotografía, se ha tomado la decisión de que el modo en que se visualice la información sea en *portrait*.

#### **1.1.5. Roles en la aplicación**

##### **Cliente**

Los clientes son aquellas personas que utilizarán la aplicación. Estos serán los que descargarán e instalarán la aplicación en su teléfono móvil. Utilizarán la aplicación para buscar artículos por palabras clave, agregarlos al carrito, y así obtener cupones de descuento y usarlos a futuro.

##### **Empresas**

Serán las empresas proveedoras de precio de los artículos, las cuales el servicio de nuestra app consultará. Para esta versión de la aplicación, los proveedores de precio serán administrados por los desarrolladores de la aplicación.

### **1.2. Alcance**

El alcance de la aplicación está dado por las siguientes características:

En primera instancia, la aplicación tendrá un solo rol de cara a los usuarios, debido a que la misma se centra en cualquier persona residente en la ciudad de Montevideo, Uruguay, y no busca distinguir roles o promociones diferenciadas.

Todos los usuarios podrán acceder a distintas funcionalidades. Las mismas incluyen funcionalidades básicas como pueden ser las siguientes:

- Visualización de la pantalla de inicio.
- Registro, Login y Logout.

Por otro lado, el usuario podrá acceder a distintas funcionalidades relacionadas con el negocio, como serán:

- Alerta de promociones de los distintos locales.
- Buscar productos en los distintos locales, ya sea por nombre o código de barras.
- La búsqueda de productos obtenida, podrá ser ordenada por ciertos criterios, como pueden ser menor precio, mayor relevancia, entre otros.
- Filtrado de productos según determinadas categorías.
- Poder marcar productos como favoritos para facilitar futuras búsquedas de productos.

- Obtener cupones de descuentos (en forma de código QR) de compras por utilizar la aplicación.
- Realizar un carrito de compras y obtener el mejor lugar para realizar la compra del carrito establecido.

Las funcionalidades destacadas contribuyen al alcance de la aplicación. El mismo fue ideado de manera alineada a los objetivos del proyecto. El alcance mencionado debe ser dividido en dos entregas las cuales son determinadas por la cátedra de Ingeniería de Software. Para la primera de ellas, se debe de entregar documentación relativa al proceso de Ingeniería de Software que utilizaremos en el transcurso del proyecto, el cual aporta valor al cliente además de ser un requisito de la materia. Finalmente, para la segunda entrega se implementarán las funcionalidades mencionadas.

### **1.3. Especificación de historias de usuario**

Las historias de usuario han sido escritas en lenguaje natural y sencillo, logrando así un detalle de la idea, el cual permite interpretarlo de forma clara, dejando margen para la discusión y profundización de la misma. Permitiendo así cumplir la regla de las 3C.

A su vez, las historias han sido escritas de forma que cumplan con las propiedades I.N.V.E.S.T, permitiendo así lograr que el trabajo sea más fluido, evitando problemas de dependencia entre historias.

#### **1.3.1. Visualizar pantalla de inicio**

##### **Especificación**

Como usuario

Quiero poder visualizar la pantalla de inicio

Para enterarme de las promociones y ofertas del día.

##### **Criterios de aceptación**

- **CA01:** Un usuario que ingrese al sistema, va a visualizar las ofertas y promociones del día (aquellos que tienen algún descuento con respecto al precio de lista). En caso de encontrarse logueado, podrá agregar los productos que desee al carrito.

**Tamaño:** 10 SP.

**Boceto de interfaz:**

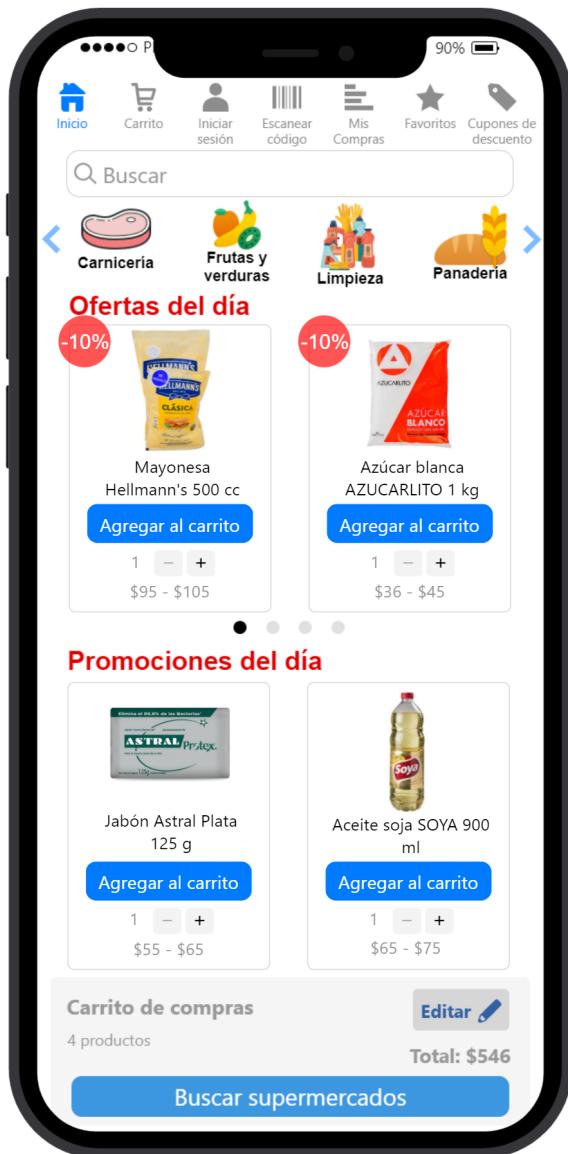


Figura 1.1: Boceto de pantalla de inicio.

### 1.3.2. Registro

#### Especificación

Como usuario

Quiero poder registrarme al sistema

Para hacer uso del mismo y obtener descuentos.

#### Criterios de aceptación

- **CA01:** Un usuario (aún no registrado) ingresa sus datos (nombre, apellido, nombre de usuario y password) correctamente y el sistema le permite registrarse.
- **CA02:** Un usuario que intente registrarse con un nombre de usuario ya registrado en el sistema, una vez ingresado los datos, se le muestra un mensaje de

error indicando que dicho nombre de usuario ya ha sido utilizado.

- **CA03:** A un usuario logueado en el sistema, no se le muestra la opción de registrarse en el mismo.

**Tamaño:** 6 SP.

**Boceto de interfaz:**

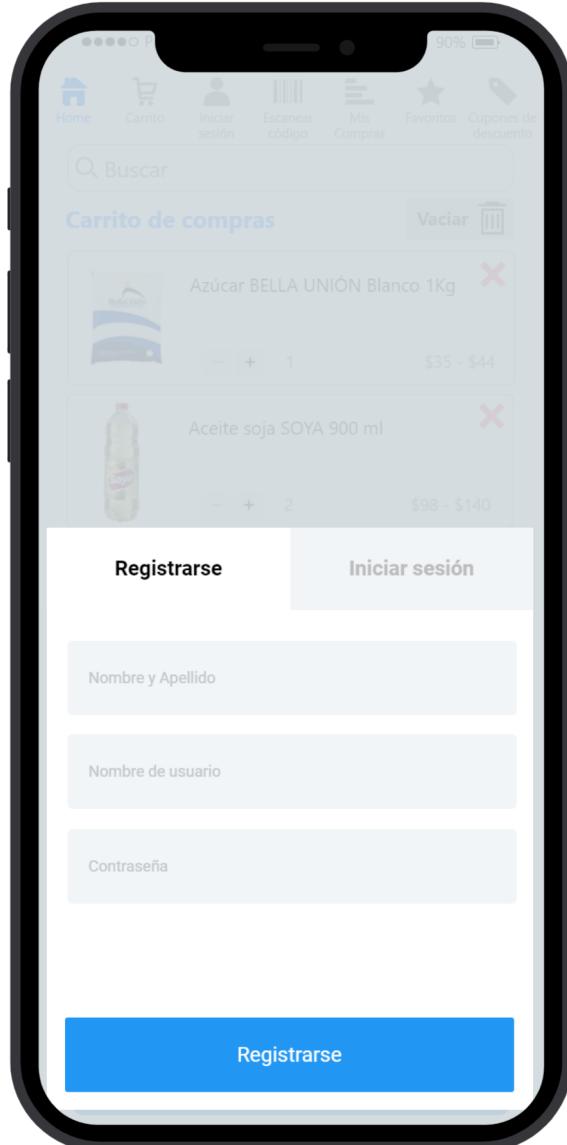


Figura 1.2: Boceto de pantalla de registro de usuario.

### 1.3.3. Alerta de promociones

#### Especificación

Como usuario

Quiero poder recibir alerta de promociones de productos

Para poder obtener grandes descuentos.

### Criterios de aceptación

- **CA01:** Un usuario logueado que se encuentre utilizando su *smartphone*, recibirá una *Push Notification* indicando el producto que se encuentra en descuento, su precio y el supermercado en el cual se encuentra la oferta. El usuario recibirá la alerta siempre y cuando el descuento sea mayor o igual al 20 % con respecto al precio de lista del producto.
- **CA02:** Un usuario que no se encuentre logueado, que se encuentre utilizando su *smartphone*, **no** recibirá una *Push Notification* indicando descuentos en ciertos productos.

**Tamaño:** 4 SP.

### Boceto de interfaz:

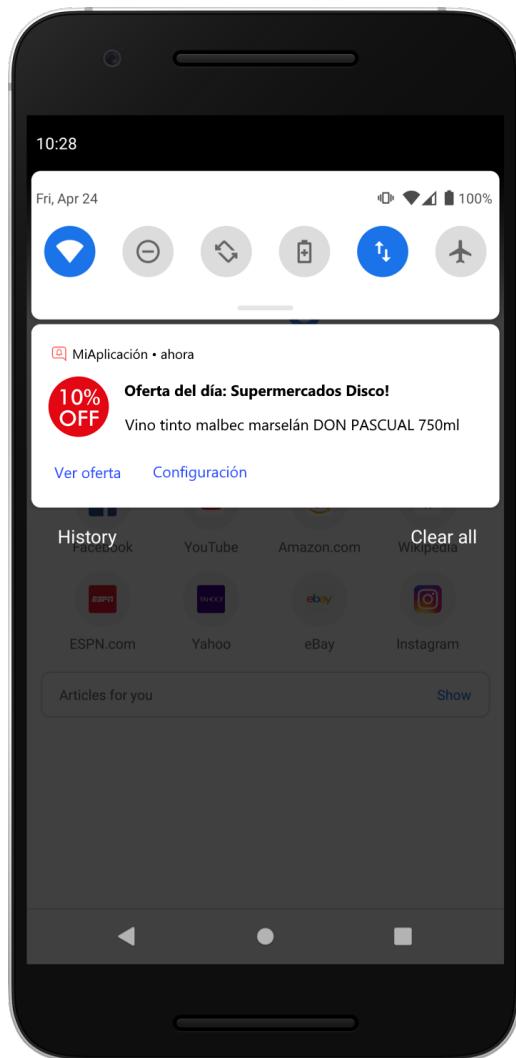


Figura 1.3: Boceto de push notification.

#### 1.3.4. Login

##### Especificación

Como usuario

Quiero poder loguearme al sistema

Para que pueda ser identificado.

##### Criterios de aceptación

- **CA01:** Un usuario (registrado al sistema) que ingresa sus credenciales correctamente (Usuario y Password) puede acceder al sistema.
- **CA02:** Un usuario (registrado al sistema) que ingresa sus credenciales incorrectamente (Usuario y Password) o están en blanco, el sistema despliega un mensaje de error y vuelve a pedir las credenciales.
- **CA03:** Un usuario que no se encuentra registrado en el sistema, al ingresar un usuario y contraseña, el sistema indica que dicho usuario no se encuentra registrado.

Tamaño: 8 SP.

##### Boceto de interfaz:

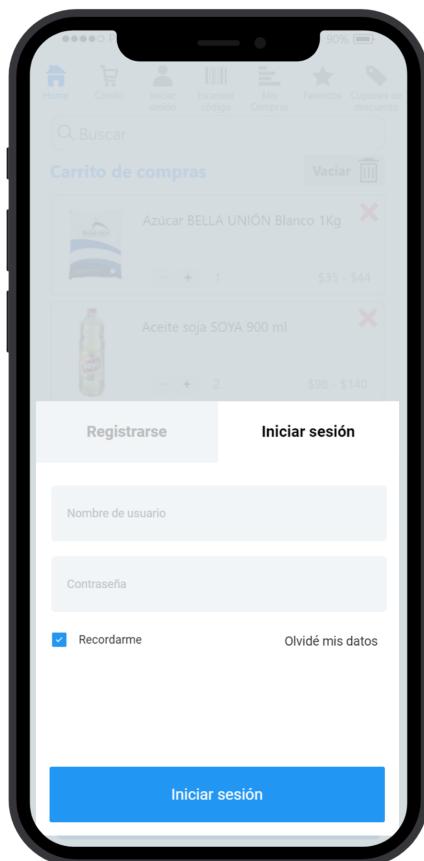


Figura 1.4: Boceto de pantalla de inicio de sesión.

### 1.3.5. Logout

#### Especificación

Como usuario

Quiero cerrar sesión en el sistema

Para asegurar que nadie indebido utilice una cuenta que no corresponde.

#### Criterios de aceptación

- **CA01:** Un usuario logueado en el sistema que presiona en el botón *salir*, debe desloguearse del sistema, y al ingresar la próxima vez deberá introducir sus credenciales.
- **CA02:** Un usuario **no** logueado, no visualizará la opción de salir del sistema.

Tamaño: 2 SP.

#### Boceto de interfaz:

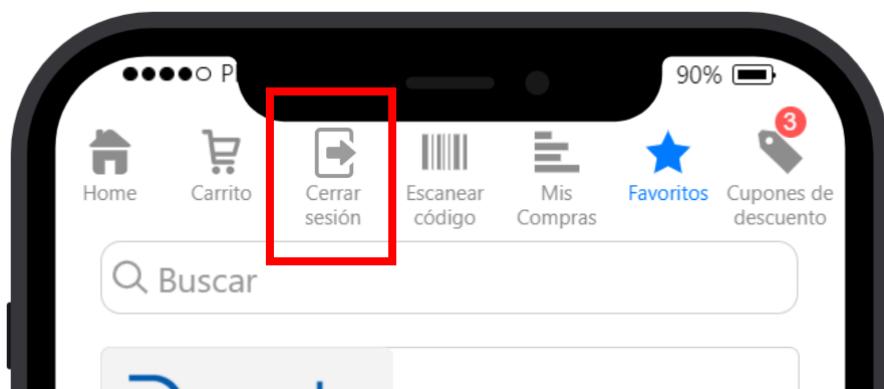


Figura 1.5: Boceto de botón de cerrar sesión.

### 1.3.6. Búsqueda mediante la barra de navegación

#### Especificación

Como usuario

Quiero poder realizar la búsqueda de un producto

Para poder visualizarlo y agregarlo al carrito.

#### Criterios de aceptación

- **CA01:** Un usuario logueado podrá utilizar la barra buscadora para encontrar productos que coincidan con su búsqueda. Una vez realizada, le aparecerá en la pantalla los productos que coincidan con la misma, permitiendo así agregarlo al carrito debido a que se encuentra logueado.
- **CA02:** Un usuario que no se encuentre logueado en la aplicación, podrá utilizar la barra buscadora para encontrar productos que coincidan con su búsqueda.

Una vez realizada, le aparecerá en la pantalla los productos que coincidan con la misma. **No** podrá agregarlo/s al carrito debido a que no se encuentra logueado.

- **CA03:** Cualquier usuario que realice la búsqueda mediante la barra buscadora, si ningún producto de los existentes en el sistema coincide con lo buscado, se mostrará un mensaje de alerta indicando que no se han encontrado resultados para la búsqueda realizada.

**Tamaño:** 3 SP.

**Boceto de interfaz:**

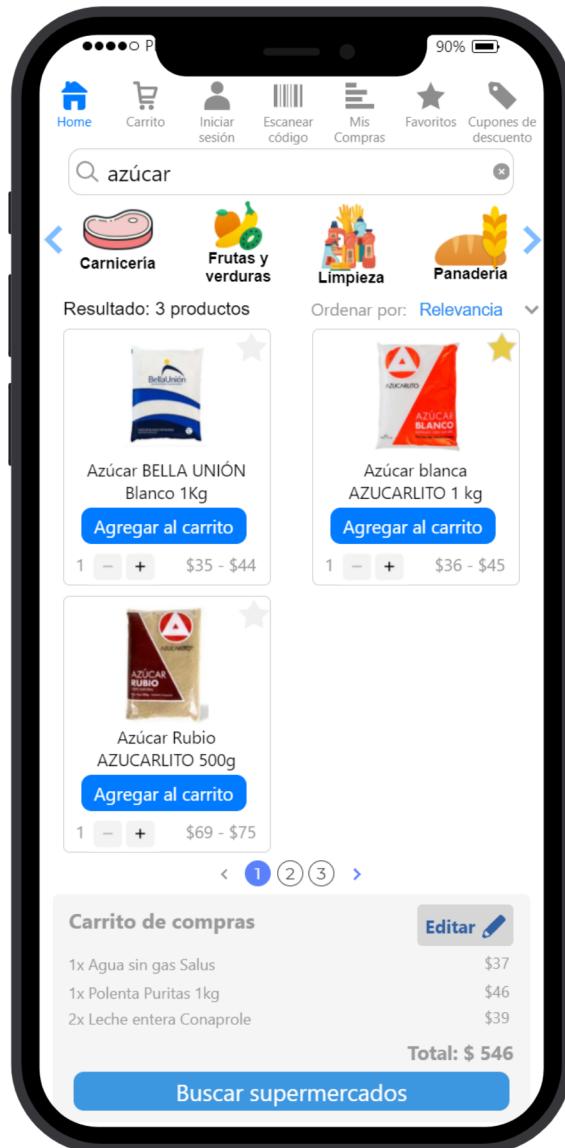


Figura 1.6: Boceto de pantalla de búsqueda de productos.

### 1.3.7. Búsqueda mediante código de barras

#### Especificación

Como usuario

Quiero poder realizar la búsqueda de un producto mediante el código de barras  
Para poder agilizar el proceso de búsqueda.

#### Criterios de aceptación

- **CA01:** Un usuario logueado podrá realizar la búsqueda mediante el código de barras (utilizando la cámara del teléfono móvil para captar el código de barras del producto buscado), permitiéndole encontrar de manera más fácil el producto buscado, mostrándole inmediatamente las especificaciones del mismo, esto ya que el código de barras es un código único, por lo cual no existirá un producto con el mismo código de barras que otro. Permitiendo así agregarlo al carrito debido a que se encuentra logueado.
- **CA02:** Un usuario que no se encuentre logueado podrá realizar la búsqueda mediante el código de barras (utilizando la cámara del teléfono móvil para captar el código de barras del producto buscado), permitiéndole encontrar de manera mas fácil el producto buscado, mostrándole inmediatamente las especificaciones del mismo, esto ya que el código de barras es un código único, por lo cual no existirá un producto con el mismo código de barras que otro. **No** podrá agregarlo al carrito debido a que **no** se encuentra logueado.
- **CA03:** Cualquier usuario que realice la búsqueda mediante el código de barras (utilizando la cámara del teléfono móvil para captar el código de barras del producto buscado), si dicho producto no existe dentro del sistema, se le mostrará un mensaje de alerta indicando que dicho producto no existe dentro del sistema.

Tamaño: 2 SP.

#### Boceto de interfaz:

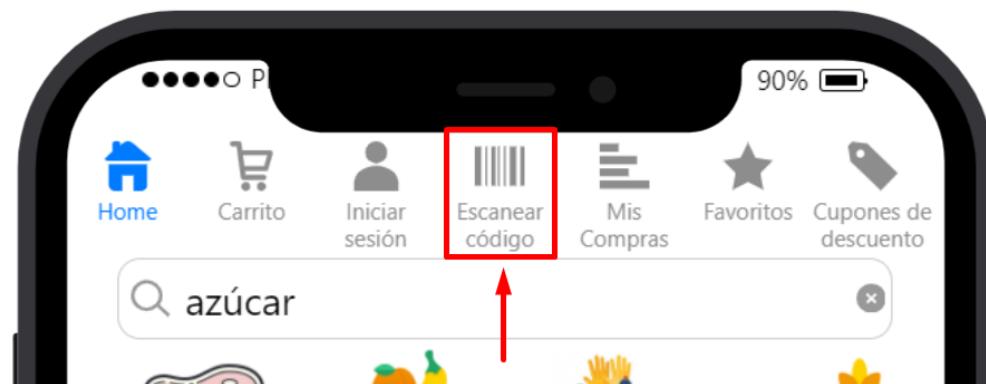


Figura 1.7: Boceto de botón para escanear código de barras.

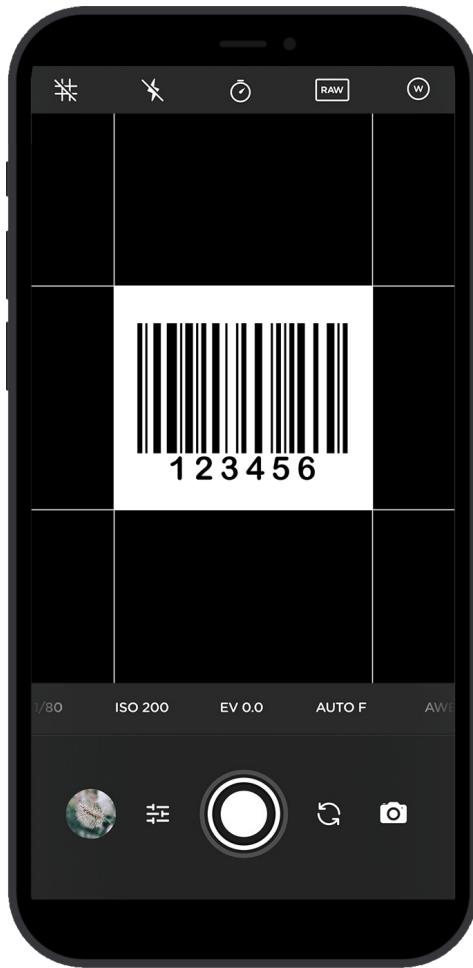


Figura 1.8: Boceto de cámara escáner de barras.

### 1.3.8. Ordenar búsqueda por criterios

#### Especificación

Como usuario

Quiero poder ordenar una búsqueda realizada mediante la barra de navegación  
Para poder visualizar productos ordenados según diferentes criterios.

#### Criterios de aceptación

- **CA01:** Un usuario logueado que haya realizado una búsqueda la cual haya obtenido resultados, podrá realizar el ordenamiento de los mismos según los siguientes criterios:
  - Relevancia (cantidad de veces que vendieron el producto).
  - Menor precio.
  - Mayor precio.
  - Alfabético.

Permitiendo así agregar al carrito el producto que considere, ya que se encuentra logueado.

- **CA02:** Un usuario que no se encuentre logueado en la aplicación, que haya realizado una búsqueda la cual haya obtenido resultados, podrá realizar el ordenamiento de los resultados obtenidos según los siguientes criterios:

- Relevancia (cantidad de veces que vendieron el producto).
- Menor precio.
- Mayor precio.
- Alfabético.

No podrá agregarlo al carrito debido a que no se encuentra logueado.

**Tamaño:** 3 SP.

**Boceto de interfaz:**

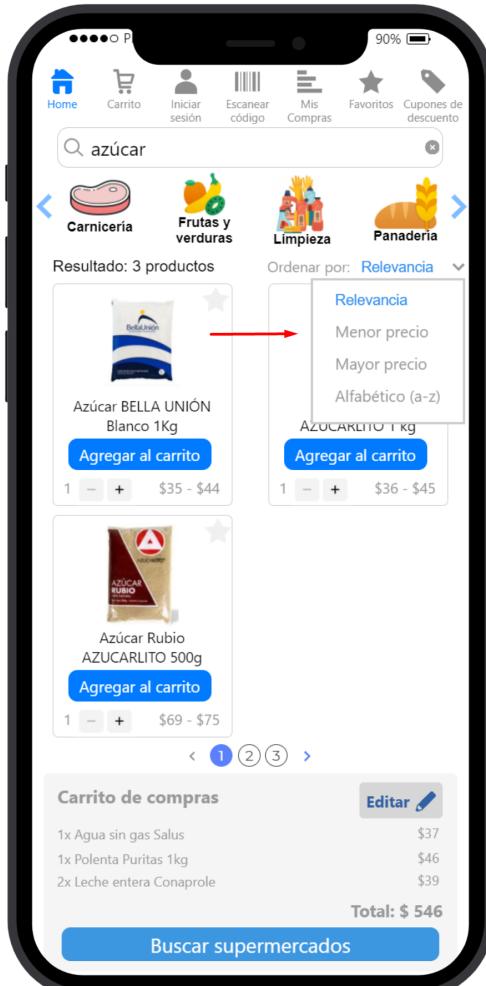


Figura 1.9: Boceto de pantalla de búsqueda con botón de ordenar productos por diferentes criterios.

### 1.3.9. Filtrado de productos según categoría

#### Especificación

Como usuario

Quiero poder filtrar la búsqueda de productos

Para poder visualizar únicamente los productos pertenecientes a una determinada categoría y encontrarlos fácilmente.

#### Criterios de aceptación

- **CA01:** Un usuario logueado podrá activar un filtro de búsqueda seleccionándolo debajo de la barra buscadora. Una vez activo el filtro, podrá ver los productos pertenecientes a la categoría seleccionada.
- **CA02:** Un usuario que no se encuentre logueado en la aplicación, podrá activar un filtro de búsqueda seleccionándolo debajo de la barra buscadora. Cuando el usuario intente agregar un producto al carrito, **no** se le permitirá, debido a que el mismo **no** se encuentra logueado.
- **CA03:** Un usuario podrá activar un filtro de búsqueda seleccionándolo debajo de la barra buscadora. En caso de que no haya ningún producto para el filtro seleccionado, se mostrará un mensaje indicando de que no hay productos disponibles para la categoría seleccionada.

Tamaño: 4 SP.

#### Boceto de interfaz:

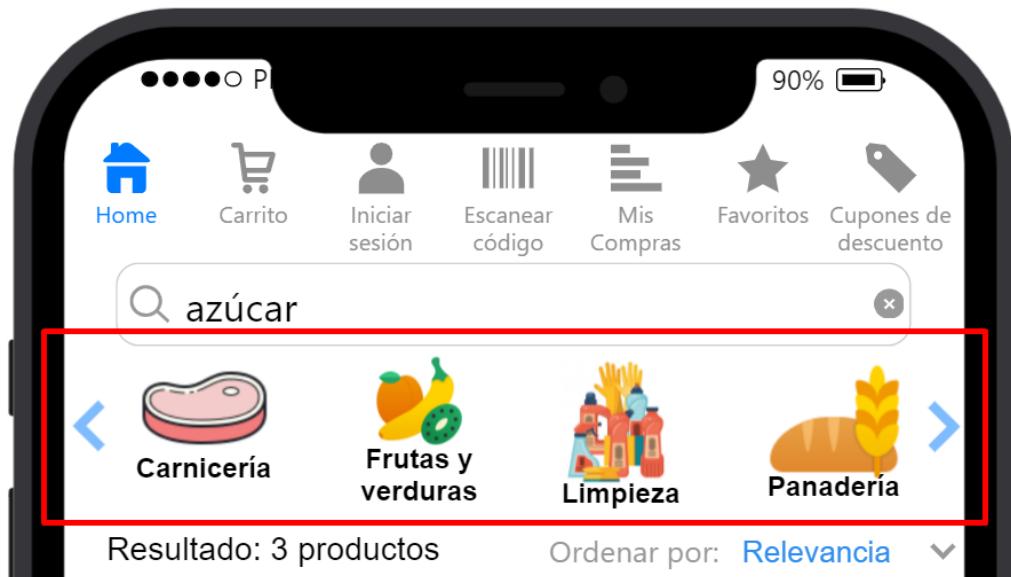


Figura 1.10: Boceto de pantalla de búsqueda con botones de categorías.

### 1.3.10. Productos favoritos

#### Especificación

Como usuario

Quiero poder marcar como favorito a un producto

Para poder hacer más sencilla la búsqueda.

#### Criterios de aceptación

- **CA01:** Un usuario logueado al realizar una búsqueda, podrá agregar los productos que desee como favoritos haciendo que el tiempo de uso de la aplicación sea menor a la hora de realizar futuras compras, comparado con un usuario que no tiene dichos productos marcados en favorito. Estos productos, se podrán visualizar en la sección "Mis favoritos".
- **CA02:** Un usuario logueado que **no** haya marcado productos favoritos, a la hora de querer visualizarlos, dirigiéndose a la sección "Mis favoritos", se le mostrará en la pantalla un mensaje indicando que no tiene productos favoritos.
- **CA03:** Un usuario que no se encuentre logueado en la aplicación, **no** podrá hacer uso de la sección "Mis favoritos" de la misma.

**Tamaño:** 3 SP.

## Boceto de interfaz:

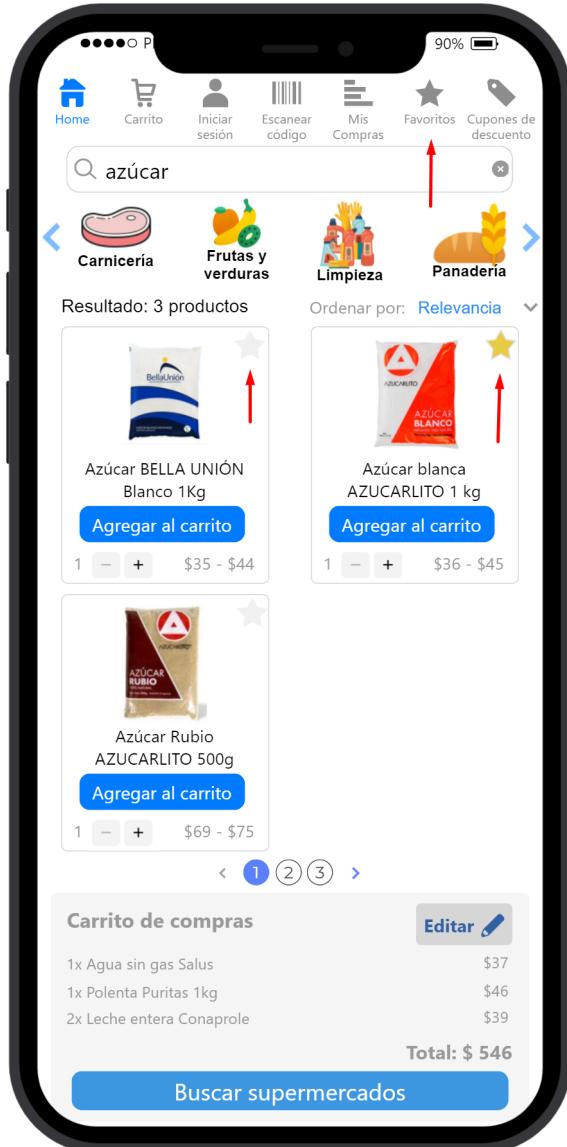


Figura 1.11: Boceto de pantalla de búsqueda con botones para agregar productos a favoritos.

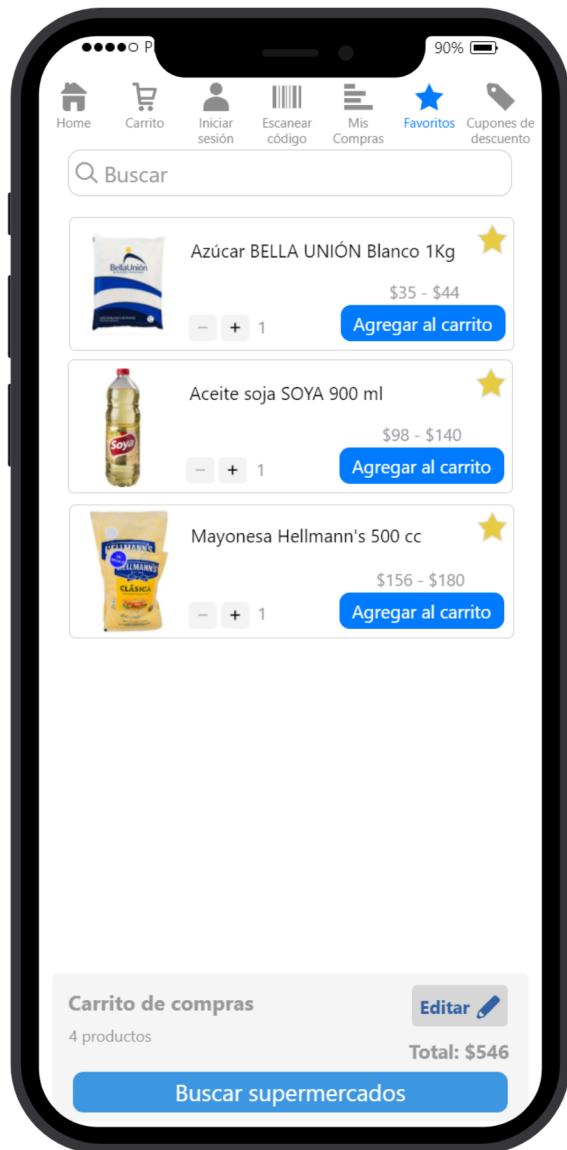


Figura 1.12: Boceto de pantalla de productos favoritos.

### 1.3.11. Cupones de descuento en compras

#### Especificación

Como usuario

Quiero poder recibir cupones de descuento

Para obtener mayores ahorros en futuras compras.

#### Criterios de aceptación

- **CA01:** Un usuario logueado tendrá el beneficio que por cada 1000\$ (pesos uruguayos) en compras se le generará un cupón de descuento con valor 50\$ (pesos uruguayos) para utilizarlo en cualquiera de los comercios adheridos (generado con un código QR). Quedando este código almacenado en la sección 'Mis cupones' a utilizar en un plazo máximo de un mes a partir de la fecha en

que se otorgó.

- **CA02:** Un usuario logueado que **no** haya realizado compra alguna, al ingresar a la sección 'Mis cupones', se le mostrará una alerta que indica que **no** tiene cupones.
- **CA03:** Un usuario que **no** se encuentre logueado en la aplicación, **no** podrá hacer uso de la sección 'Mis cupones' de la misma.
- **CA04:** Un usuario logueado que haya realizado compras haciendo uso de la aplicación, no obtendrá un cupón de descuento hasta que el total acumulado de gastos ascienda los 1000\$ (pesos uruguayos). Siendo así que por cada 1000\$ (pesos uruguayos) ocurre lo descrito en el criterio de aceptación 01.

**Tamaño:** 3 SP.

**Boceto de interfaz:**

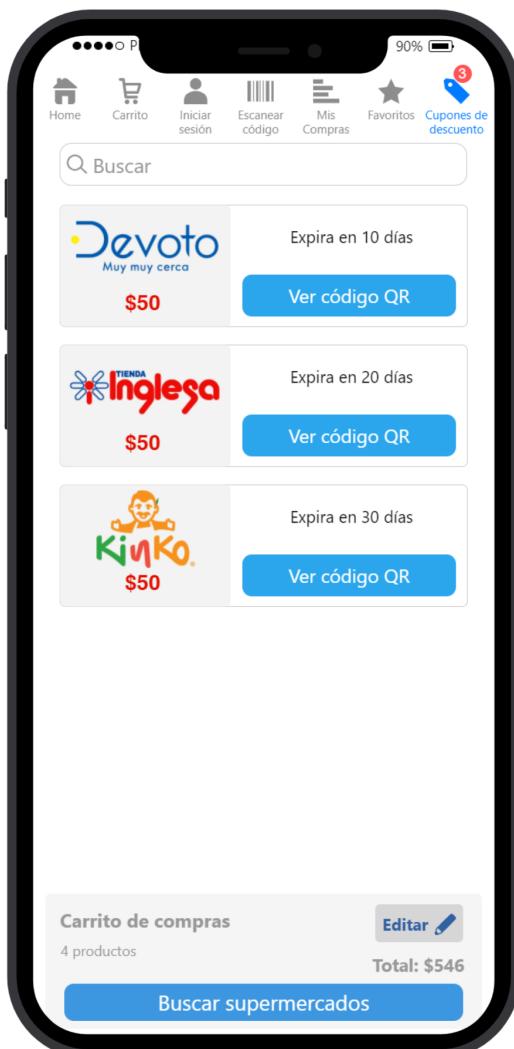


Figura 1.13: Boceto de pantalla de cupones.

### **1.3.12. Visualizar carrito de compras**

#### **Especificación**

Como usuario

Quiero poder visualizar los productos del carrito de compras

Para conocer los productos adquiridos hasta el momento y poder agregar o disminuir la cantidad comprada de cada uno de ellos.

#### **Criterios de aceptación**

- **CA01:** Un usuario logueado que haya agregado algún producto para adquirir, podrá presionar el ícono identificado por un carrito, y obtener los productos que ha adquirido hasta ese momento (nombre, unidades y precio). En esta opción, tendrá disponible un botón identificado por un símbolo de '+' que permite incrementar las unidades de ese producto, así como un símbolo de '-' que permite decrementar las unidades adquiridas al momento.
- **CA02:** Un usuario que no se encuentre logueado en el sistema, al presionar el ícono identificado por un carrito, se lo redireccionará a la pantalla de *Login* para que se autentique en el sistema.
- **CA03:** Un usuario logueado que no haya adquirido producto alguno, al presionar el ícono del carrito se le mostrara una alerta indicando: "No ha agregado productos al carrito".

**Tamaño:** 5 SP.

#### **Boceto de interfaz:**

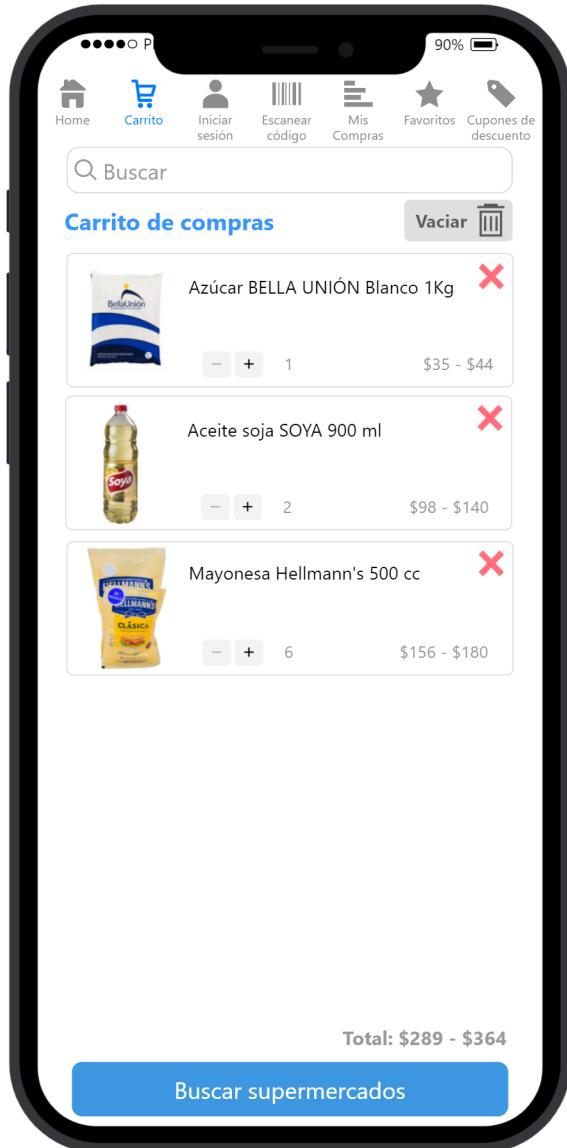


Figura 1.14: Boceto de pantalla de carrito.

### 1.3.13. Obtener la mejor opción de compra para un conjunto de productos

#### Especificación

Como usuario

Quiero poder obtener el precio total en cada uno de los supermercados para un conjunto de productos seleccionados

Para obtener la mejor opción considerando: precio y distancia.

#### Criterios de aceptación

- **CA01:** Un usuario logueado que haya seleccionado al menos un producto previamente, al encontrarse en la sección 'Carrito', al presionar el botón "Buscar supermercados", podrá visualizar un mapa (con la distancia a cada supermer-

cado y el precio para el conjunto de productos del carrito), y debajo, un listado con los supermercados que ofrecen los productos seleccionados incluyendo para cada uno de ellos: el precio, la distancia a la que se encuentra el supermercado que lo ofrece y el tiempo que resta para que el supermercado cierre.

En la lista de opciones, aparecerá en primer lugar, indicándose como opción recomendada, el supermercado que tenga la mejor oferta para el conjunto de productos seleccionados. Se entiende que el supermercado que presenta la mejor oferta, es aquel en el que el valor de la suma de los precios de los productos seleccionados es el menor.

En el caso de que dos o más supermercados tengan el mismo precio para el conjunto de productos seleccionados, se determinará como opción recomendada al que se encuentre más próximo al usuario considerando la localización del mismo al momento de hacer uso de la aplicación.

En caso de que la igualdad persista, es decir, que ambas opciones se encuentren a la misma distancia, se considerara el parámetro horario, indicando como opción recomendada al supermercado que posee el horario más extendido.

- **CA02:** Un usuario logueado que no haya seleccionado al menos un producto previamente, luego de presionar el botón identificado por un carrito, le mostrará una alerta de que no hay productos en el carrito, por lo cual no podrá visualizar ningún supermercado (debido a que no ha seleccionado nada previamente).

**Tamaño:** 5 SP.

**Boceto de interfaz:**

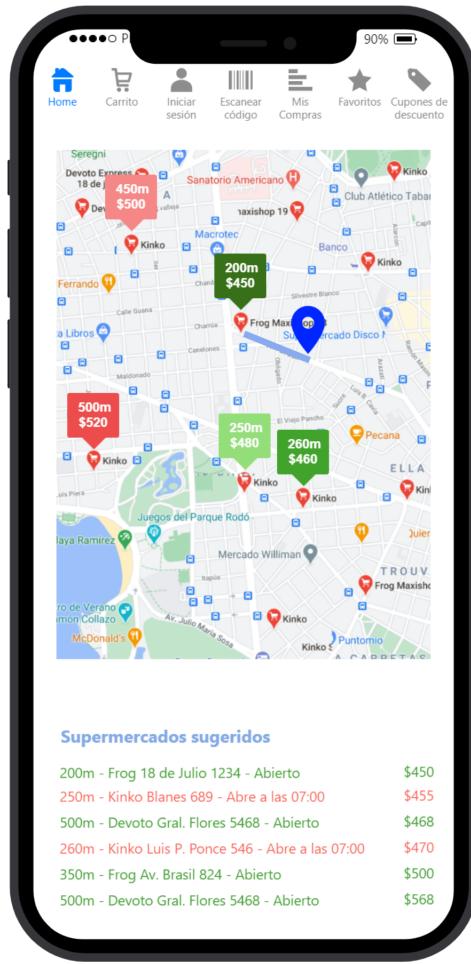


Figura 1.15: Boceto de pantalla de supermercados sugeridos.

## 1.4. Dispositivos del celular utilizados

Para este proyecto se presentan funcionalidades que requieren el uso de ciertos dispositivos que hacen que esta aplicación deba ser ejecutada en un celular.

### GPS

Este dispositivo debe ser utilizado para poder localizar la ubicación del dispositivo. Con dicha ubicación y utilizando la API de Google Maps se logrará mostrar donde se ubican los diferentes supermercados donde se podrá realizar la compra o adquirir los productos del carrito. Para esto, es necesario pedir los siguientes permisos:

- INTERNET
- ACCESS\_FINE\_LOCATION
- ACCESS\_COARSE\_LOCATION

## Cámara

La cámara del celular es necesaria para una de las funcionalidades planteadas. Esta consiste en la capacidad de poder escanear un códigos de barras para realizar la búsqueda de un producto. Para esto es necesario el siguiente permiso:

- CAMERA

## 2. Gestión de proyecto

### 2.1. *Metodología*

Para decidir con qué enfoque se llevará a cabo este proyecto, se realizó un análisis de las dos alternativas propuestas (basado en planes o en Metodologías Ágiles) llegando a la conclusión de que un enfoque basado en metodologías ágiles es el adecuado para este proyecto. Donde las razones que influyeron para tomar esta decisión son las siguientes:

- Tiempo y entrega de valor: se cuenta con un tiempo limitado para desarrollar el sistema. Donde toda la documentación y planificación que conlleva un enfoque basado en planes reduciría tiempo valioso para realizar entregas tempranas de valor al cliente. Haciendo énfasis en la entrega de valor, con un enfoque ágil se lograrían entregas frecuentes y tempranas de valor.
- Falta de conocimiento en el área y retroalimentación: al no contar con conocimientos sólidos de las tecnologías a utilizar además de desconocer las capacidades del equipo, se da entender, la dificultad que conlleva realizar una planificación rigurosa y realista del proyecto en un enfoque basado en planes. A su vez, al no conocer claramente el mercado al que se ingresa con el sistema a desarrollar será muy útil la entrega temprana de valor que nos permitirá obtener retroalimentación.

### 2.2. *Scrum*

Dada la naturaleza del proyecto, el equipo se inclinó por seguir una metodología ágil basada en *Scrum*.

El principal motivo que nos llevó a decantarnos por la metodología mencionada, es que nos enfrentamos al desarrollo de un sistema que requiere el uso de tecnologías totalmente desconocidas para el equipo de desarrollo (*Kotlin*, desarrollo *mobile*), que requiere un importante grado de esfuerzo e investigación por parte de los miembros del equipo y con requerimientos que pueden verse afectados a lo largo del proyecto.

Por lo mencionado, es que decidimos poner foco en una metodología como *Scrum* que permita de cierto modo esa flexibilidad que necesitamos para poder investigar acerca de las tecnologías de manera libre sin estar atado a un plan estricto, minimizar

los riesgos que puedan presentarse y adaptar los requisitos en caso de que fuera necesario.

## 2.2.1. Ceremonias

### ***Sprint Planning***

En esta reunión, se indica que trabajo se debe realizar durante el *Sprint*. La planificación del *Sprint* consta de los siguientes temas:

- Se propone un aumento en el valor del producto y utilidad en el sprint actual.  
En esta primera parte se define el *Sprint Goal*.
- Que se puede hacer este sprint: Se seleccionan los elementos del *Product Backlog*.

### ***Daily Scrum***

Estas reuniones se realizarán de forma diaria (mediante videollamada) con un tiempo límite de 15 minutos, para que sea breve y se mencionen puntos importantes.

Durante el *daily scrum*, cada miembro del equipo responde las siguientes 3 preguntas:

- ¿Qué hiciste ayer?
- ¿Que harás hoy?
- ¿Hay impedimentos en tu camino?

Al enfocarse en lo que cada miembro del equipo hizo ayer y hará hoy, el equipo gana una visión general de lo que se ha realizado y aquello que falta por realizar.

### ***Sprint Review***

Al finalizar el *sprint*, el equipo hará una revisión para inspeccionar el incremento y adaptar el *Product Backlog* en caso de que fuese necesario. En esta revisión, se analiza lo hecho y las modificaciones en el *Product Backlog* que se dieron en el *sprint*. Según lo sugerido por Scrum, esta reunión a menudo no debe durar más de cuatro horas, en nuestro caso, consideramos que no debería extenderse por más de 1.5 - 2 horas considerando el alcance de cada *sprint*. Esta reunión da como resultado un *Product Backlog* que se ha revisado y se utilizará para el siguiente *sprint* (que hacer a continuación). Se puede ajustar el trabajo pendiente.

### ***Sprint Retrospective***

El *Sprint Retrospective* es una oportunidad para el Equipo *Scrum* de inspeccionarse a sí mismo y de crear un plan de mejoras que sean abordadas durante el siguiente *sprint*.

Básicamente los aspectos a desarrollar en esta reunión son los siguientes:

- Inspeccionar cómo fue el último *sprint* en cuanto a personas, relaciones, procesos y herramientas.
- Identificar y ordenar los elementos más importantes que salieron bien y las posibles mejoras.
- Crear un plan para implementar las mejoras a la forma en la que el Equipo *Scrum* desempeña su trabajo.

### **2.2.2. Roles**

Sea este documento un reflejo del obligatorio integrado por 3 personas, se realiza una adaptación de los roles típicos de *Scrum*:

- *Development Team*: Joaquín, Agustín y Santiago. Siendo programadores, analistas, analistas de calidad, *testers*, etc.
- *Scrum Master*: Es el responsable de asegurar que *Scrum* sea bien entendido y adoptado. Este rol lo llevará a cabo Joaquín, apoyándose en los consejos de los docentes.
- *Product Owner*: En conjunto con los interesados (docentes del curso), determinar prioridades y alcance.

### **2.2.3. Product Backlog**

Listado ordenado y priorizado de los requisitos necesarios para la implementación de un proyecto. El cual contiene las descripciones de las funcionalidades y características que se quieran en el producto, ordenadas en función de su prioridad.

#### ***Management Product Backlog***

La creación y organización del *Product Backlog* es crucial, dado que hay que tener en cuenta los riesgos explicados en la sección Gestión de Riesgos. Para ello se organizarán y asignarán las tareas de cada sprint con sumo cuidado de no generar un atraso en el proceso. Concluyendo así la necesidad de la existencia de una reunión semanal con una duración máxima de una hora para fijar el alcance de cada sprint con los riesgos conocidos.

#### ***GitHub board***

Esta herramienta fue la elegida para almacenar las tareas y US, además de realizar la asignación y ser el lugar de seguimiento del proyecto. El mismo presenta las columnas *Backlog* con las US, *Done Stories*, con las US terminadas hasta el momento del desarrollo del proyecto y *Sprint Backlog* con las US especificadas para el *sprint* en el cual se encuentra el equipo de desarrollo. Por otro lado, tendremos las columnas *To Do*, *Doing* y *Done*, las cuales son utilizadas en el *sprint* que el equipo se encuentre realizando para marcar el estado de las tareas en el mismo. Estas, permitirán al equipo saber que tareas han seleccionado los miembros del equipo.

## 2.2.4. Definition of Done

La *Definition of Done*, es un entendimiento compartido de lo que significa que una tarea está terminada. Nos parece oportuno definirlo de antemano para evitar confusiones y ambigüedades al verificar si una tarea está terminada o no.

- Se considerara una historia como terminada cuando el equipo se reúne a realizar la integración, los componentes funcionan correctamente y todos los integrantes del equipo están de acuerdo con la implementación realizada.
- Los flujos alternativos deben estar contemplados, implementados y testeados.
- Todas las pruebas unitarias deben pasar de forma satisfactoria, y el porcentaje de cobertura debe ser de al menos el 80 %.
- Antes de realizar un *merge* a la rama *develop*, el desarrollador deberá verificar que se cumplen los criterios aquí establecidos.

## 2.3. Release plan

*Release planning* (planificación del *release* o lanzamiento) es el proceso de crear un plan de alto nivel que cubra un período mayor a una iteración. Un *release* típicamente abarca de 3 a 6 meses y entre 3 y 12 o más iteraciones, dependiendo del largo de las iteraciones. En este caso los *releases* van a ser de períodos más cortos ya que los tiempos de entrega ya están determinados y hace imposible abarcar *releases* de un largo típico.

Un plan de *release* es importante porque:

- Ayuda al PO y a todo el equipo a decidir cuánto hay que desarrollar y cuánto tiempo llevará hasta tener un producto 'lanzable' y poder hacer valer la inversión.
- Un *release* conlleva expectativas sobre qué se tiene que desarrollar en qué período de tiempo. Es útil para otras actividades estratégicas de la organización.
- Sirve como punto de referencia hacia el cual el equipo va a progresar. Sin el concepto de *release*, los equipos van de iteración en iteración sin fin. Provee un contexto que le permite al equipo saber dónde se encuentran con respecto al objetivo final.

### 2.3.1. Condiciones de satisfacción

El éxito del proyecto estará determinado por la entrega de los requerimientos definidos mediante las historias de usuario en los plazos de tiempo que se definirán.

### 2.3.2. Estimación de historias de usuario

El proyecto constará de un único *release* que tiene como fecha de inicio el 15/04/2021 y fecha de finalización el 16/06/2021. Si bien la instancia de entrega de documentación del proyecto tiene como plazo máximo de entrega el día 21/04/2021, estimamos que, dado el grado de avance de la misma, es posible finalizarla antes, y comenzar con lo que es el desarrollo de la aplicación el día 15/04/2021.

Primer (y único) *release*: 15/04/2021 - 16/06/2021 (*Date Driven*).

Que el *release* sea de tipo *Date Driven*, significa que se debe determinar que se puede hacer antes de una fecha. Esto es útil para saber si el proyecto va en acorde con los objetivos de la organización.

Para estimar las historias de usuario, utilizamos la técnica de *Planning Poker*, que consiste en calcular una estimación basada en el consenso, utilizada para estimar el esfuerzo o el tamaño relativo de las historias de usuario en el desarrollo de software. Es decir, cada uno estima el valor en puntos de historia, de forma relativa (es decir, teniendo en cuenta los valores que se estimaron para otras historias), y luego se discute en conjunto para llegar a un valor de estimación.

Historia de usuario	Tamaño (SP)
1.2.1 Visualizar pantalla de inicio	10
1.2.2 Registro	6
1.2.3 Alerta de promociones	4
1.2.4 Login	8
1.2.5 Logout	2
1.2.6 Búsqueda mediante barra de navegación	3
1.2.7 Búsqueda mediante código de barras	2
1.2.8 Ordenar búsqueda por criterios	3
1.2.9 Filtrado de productos según categoría	4
1.2.10 Productos favoritos	3
1.2.11 Cupones de descuento en compras	3
1.2.12 Visualizar carrito de compras	5
1.2.13 Obtener la mejor opción de compra para un conjunto de productos	5

Tabla 2.1: Estimación en SP para cada historia de usuario

### 2.3.3. Largo de la iteración

*Sprint* es el nombre que va a recibir cada uno de los ciclos o iteraciones que vamos a tener dentro del proyecto.

Nos van a permitir tener un ritmo de trabajo con un tiempo prefijado, siendo la duración habitual de un *Sprint* unas cuatro semanas, pero dado el tiempo existente para la entrega del proyecto, **los *Sprints* tendrán una duración de una semana**, esto avalado por la metodología *Scrum* que nos dice que la duración de la iteración debería estar entre una semana y un máximo de dos meses.

En cada *Sprint* o cada ciclo de trabajo lo que vamos a conseguir es lo que se denomina un entregable o incremento del producto, que aporte valor al cliente.

Decidimos realizar iteraciones cortas, porque consideramos que esto trae ventajas, sobre todo teniendo en cuenta que se utilizarán tecnologías nuevas que los miembros del equipo desconocen.

Precisamente, para nuestro proyecto, alguna de las ventajas que presenta trabajar en iteraciones cortas, son las siguientes:

- Mejora de la comunicación entre los miembros del equipo, debido a que se acortan los ciclos de comunicación.
- Mejora el aprendizaje del equipo sobre su forma de trabajar y su cohesión como equipo.
- El equipo reacciona mucho más rápido a los cambios de dirección o estrategia.
- Al hacer *sprints* cortos, se pueden realizar revisiones de *Sprint* más frecuentes, que permiten al cliente poder realizar comentarios de forma más frecuente para mostrar su opinión con respecto a la lista de trabajos pendientes. Esto debería reducir en gran medida la necesidad de que el cliente solicite un cambio durante un *Sprint* en curso.

#### 2.3.4. Estimación de velocidad

Generalmente la estimación en metodologías ágiles, se realiza por medio del promedio de las velocidades que el equipo desarrolló en *sprints* anteriores. En nuestro caso, esa forma de estimación no es posible dado que, no hemos trabajado en conjunto los tres miembros del equipo en proyectos de desarrollo de software.

Otras razones por la que la estimación es difícil:

- El desarrollo de software es una actividad de diseño.
- La tecnología con las cuales se trabaja cambian rápidamente, esto puede producir que lleve más tiempo el desarrollo. También influye si los miembros del equipo de desarrollo han trabajado con dichas tecnologías, dado que los individuos son difíciles de predecir y cuantificar.
- Los requerimientos no son tangibles, debido a que el software no lo es, es difícil saber estimar cuánto vale una funcionalidad antes de tenerla hecha y funcionando.

- Muchas veces, a medida que se producen avances en el proyecto, como el sistema es 'blando', establece que los requerimientos pueden cambiar, lo que conduce a mayor tiempo de desarrollo.

Entonces, lo que finalmente se hizo, fue considerar la cantidad de horas y días de la semana que el equipo va a trabajar desarrollando, en nuestro caso, 4 horas diarias de lunes a sábado, lo que hace un total de 24 horas semanales. Luego, desglosamos cada una de las historias de usuario en tareas y estimamos la cantidad de horas/-persona que se tiene que destinar a cada una de ellas para finalizarlas. A partir de lo anterior, se estimó para cada historia la cantidad de horas/persona requeridas para completarla. Se definió en base a ello, que la cantidad de SP que el equipo puede realizar en una iteración es igual a **10**. En base a este número, y sabiendo la cantidad de horas/persona que habría que dedicarle a cada historia de usuario, se completó la tabla 2.3.2.

### 2.3.5. Priorización de historias de usuario

El PO se basa en la investigación del mercado para priorizar las *user stories*. La lista junto con las estimaciones en tamaño, es la presentada en la tabla 2.3.2. El orden fue determinado de forma tal de evitar las dependencias entre las historias de usuario. Como se trata de un proyecto basado en la fecha (*date-driven*), lo que sucede es que se sabe cuantas iteraciones existentes va a haber, de forma que hay tiempo para 6 iteraciones seguidas de una semana extra de ser necesario. Sabiendo que la velocidad del equipo es de 10 SP, el equipo podrá desarrollar un total de 60 SP en el *release* planificado. En la siguiente tabla, se puede observar como el total estimado en *Story Points* para el proyecto es igual a 58.

Historia de usuario	Tamaño (SP)
1.2.1 Visualizar pantalla de inicio	10
1.2.2 Registro	6
1.2.3 Alerta de promociones	4
1.2.4 Login	8
1.2.5 Logout	2
1.2.6 Búsqueda mediante barra de navegación	3
1.2.7 Búsqueda mediante código de barras	2
1.2.8 Ordenar búsqueda por criterios	3
1.2.9 Filtrado de productos según categoría	4
1.2.10 Productos favoritos	3
1.2.11 Cupones de descuento en compras	3
1.2.12 Visualizar carrito de compras	5
1.2.13 Obtener la mejor opción de compra para un conjunto de productos	5
<b>Total</b>	<b>58</b>

## 2.4. Sprint planning

Para esta primera etapa del proyecto, realizamos una primera estimación de cada uno de los *sprints* del proyecto, estimando cada una de las tareas necesarias

para cumplir con cada una de las historias de usuario asignadas en cada *sprint*. Sin embargo, durante el proyecto, como se mencionó previamente se realizaran *Sprint planning meeting* en donde, utilizando como base las planificaciones de los *sprint* que se muestran a continuación, se planificarán cada uno de los *sprints* ajustando las planificaciones iniciales, teniendo en cuenta la velocidad desarrollada por el equipo en *sprints* anteriores.

### 2.4.1. Sprint 1

#### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

#### Objetivo del sprint

Implementar de forma completa la funcionalidad que permite al usuario visualizar la pantalla de inicio con ofertas y promociones del día.

#### Historias de usuario

- 1.2.1 Visualizar pantalla de inicio.

#### División de historias de usuario en tareas

	<b>Historias de usuario</b>	<b>Tareas</b>	<b>Estimación</b>	<b>Estimación total</b>
<b>Sprint 1</b>	1.2.1 Visualizar pantalla de inicio.	Construcción del esqueleto de la interfaz dentro de la aplicación. Incluyendo el menú superior, los espacios para las ofertas y promociones, y categorías.	4 hs persona	<b>22 hs persona</b>
		Construcción del esqueleto de las ofertas del día, esto incluye el botón de agregar al carrito, las cantidades e imagen del producto	2 hs persona	
		Construcción del esqueleto de las promociones del día, esto incluye el botón de agregar al carrito, las cantidades e imagen del producto.	1 hs persona	
		Lógica para obtener ofertas y promociones.	15 hs persona	

Figura 2.1: División de historias del sprint en tareas, y estimación de las mismas.

## **Periodo de fechas**

Este sprint tiene como fecha de inicio el 15/04/2021 y como fecha de finalización el 22/04/2021.

## **2.5. Ejecución del Sprint 1**

### **2.5.1. Sprint Review**

Como se puede apreciar en la siguiente Burndown Chart, el equipo no cumplió con todo lo planificado para el sprint 1, restando un total de 5 SP que se replanificaron para completar en el siguiente sprint. Analizando lo sucedido, vemos que desde el 15/04 al 18/04 el equipo no avanzó en lo que es el desarrollo del sistema debido a que la curva de aprendizaje para la comprensión de las tecnologías a utilizar fue más compleja de lo estimado, y ese periodo de tiempo, se invirtió en estudiar documentación y materiales que permitan, a partir del 19/04 el desarrollo a mayor velocidad.

Un punto muy importante que se puede notar, es el tiempo de dedicación el cual se le destinó a la creación de interfaces de usuario. A las mismas, el equipo las cuestionó como que no afectarían en gran medida el desarrollo, poniendo en la estimación un mayor énfasis a la hora del desarrollo de la lógica para realizar la conexión entre el BackEnd y el FrontEnd, esto luego de comenzar con la correspondiente implementación se logró observar que obtener interfaces de gran calidad conlleva un mayor tiempo. Donde el énfasis es la calidad, mantenibilidad y entendimiento del código.

Siendo lo dicho anteriormente, que al poner énfasis en la calidad de las interfaces, se notó una mayor dedicación de tiempo destinado al entendimiento para realizar la creación de las mismas. Por otro lado, la correspondiente dedicación que conlleva aplicar patrones y calidad en el código, implica mayor tiempo aún. Concluyendo así, que el equipo debería haber realizado lo que se conoce como un Spike. Donde el spike es un elemento del Product Backlog orientado a la investigación o experimentación, cuya finalidad es obtener el aprendizaje necesario para implementar la funcionalidad solicitada por el Product Owner o cliente.

De forma tal que la manera correcta de haber realizado lo mencionado anteriormente, es haber representado el spike según una de las siguientes formas.

1. El elemento del Product Backlog o historia de usuario original se divide en dos: uno referido a la investigación, a la obtención de información o conocimiento adicionales, y el otro referido a la funcionalidad en sí.
2. También puede que el spike sea una tarea del Sprint Backlog, asociada al elemento del Product Backlog sobre la funcionalidad de interés.

Por lo cual para próximos sprint, lo que se nos recomendó consultando a expertos, es realizar la definición de un timebox o duración máxima para el spike, habitualmente en días. Por tanto, limitamos de antemano el tiempo que vamos a invertir aprendiendo sobre la tecnología en cuestión.

## Burndown chart Sprint 1

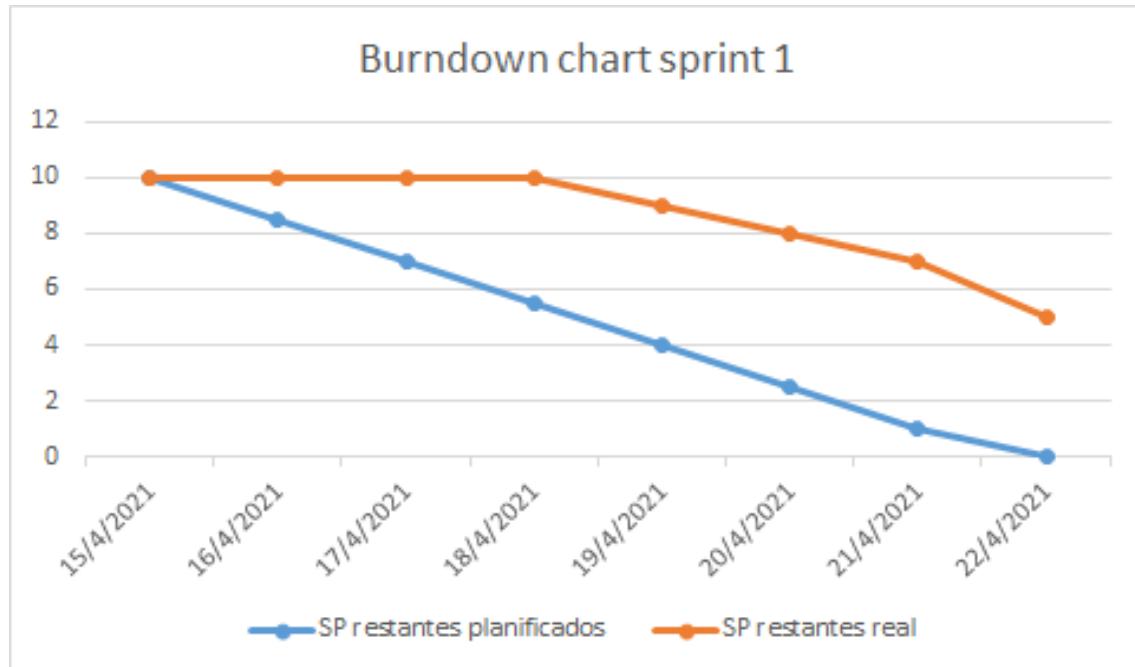


Figura 2.2: Burndown chart del Sprint 1

Fecha	SP restantes planificados	SP restantes real
15/4/2021	10	10
16/4/2021	8,5	10
17/4/2021	7	10
18/4/2021	5,5	10
19/4/2021	4	9
20/4/2021	2,5	8
21/4/2021	1	7
22/4/2021	0	5

Figura 2.3: Comparación día a día: SP restantes planificados VS SP restantes real

### 2.5.2. Sprint Retrospective

Como equipo, podemos notar que como aspecto positivo, tanto las interfaces de usuario como el código desarrollado son de calidad, ya que debido a que antes de terminar una tarea el miembro del equipo que desarrolló solicitó un *pull request*, teniendo la revisión de cada uno de los demás desarrolladores que permitieron garantizar el cumplimiento de los estándares y cobertura de pruebas acordados.

Como punto negativo, como se mencionó más arriba, no se cumplieron con todas las tareas que estaban planificadas debido a que la curva de aprendizaje fue superior a la estimada.

Como equipo, en el próximo sprint, teniendo los conocimientos adquiridos en este sprint sobre las tecnologías a utilizar, debemos incrementar el ritmo de desarrollo con respecto al planificado e incluir las tareas que no se pudieron desarrollar en este sprint.

### 2.5.3. Sprint 2 (Original)

#### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

#### Objetivo del sprint

Implementar de forma completa la funcionalidades de Registro de usuario, y alerta de promociones.

#### Historias de usuario

- 1.2.2 Registro.
- 1.2.3 Alerta de promociones.

#### División de historias de usuario en tareas

	<b>Historias de usuario</b>	<b>Tareas</b>	<b>Estimación</b>	<b>Estimación total</b>
<b>Sprint 2</b>	1.2.2 Registro.	Construcción del esqueleto, un formulario donde se solicitan los datos aclarados en la historia de usuario.	3 hs persona	<b>22 hs persona</b>
		Lógica para registrar usuarios y creación.	10 hs persona	
	1.2.3 Alerta de promociones	Construcción del esqueleto de la notificación, lo que incluye el producto, precio de oferta y supermercado donde se encuentra.	2 horas persona	
		Implementación en Firebase que permita la emisión de la alerta de notificación.	7 horas persona	

Figura 2.4: División de historias del sprint en tareas, y estimación de las mismas.

## Periodo de fechas

Este sprint tiene como fecha de inicio el 23/04/2021 y como fecha de finalización el 30/04/2021.

### 2.5.4. Sprint 2 (Revisado)

#### Velocidad a alcanzar

La velocidad a alcanzar es de 11 SP.

#### Objetivo del sprint

Terminar con la tarea 'Lógica para obtener ofertas y promociones' del sprint 1. Implementar de forma completa la funcionalidades de Registro de usuario, y alerta de promociones.

#### Historias de usuario

- 1.2.1 Visualizar pantalla de inicio.
- 1.2.2 Registro.
- 1.2.3 Alerta de promociones.

#### División de historias de usuario en tareas

	Historias de usuario	Tareas	Estimación	Estimación total
Sprint 2	1.2.1 Visualizar pantalla de inicio	Lógica para obtener ofertas y promociones	8 hs persona	24 hs persona
	1.2.2 Registro	Construcción del esqueleto, un formulario donde se solicitan los datos aclarados en la historia de usuario.	2 hs persona	
		Lógica para registrar usuarios y creación.	5 hs persona	
	1.2.3 Alerta de promociones.	Construcción del esqueleto de la notificación, lo que incluye el producto, precio de oferta y supermercado donde se encuentra.	2 hs persona	
		Implementación en Firebase que permita la emisión de la alerta de notificación.	7 hs persona	

Figura 2.5: División de historias del sprint en tareas, y estimación de las mismas.

### 2.5.5. Sprint review

Como se puede apreciar en la siguiente Burndown Chart, el equipo cumplió con notoriedad todo lo planificado para el sprint 2. En particular además de cumplir con lo planificado, sucedió con el sprint 1 que restaron 5 SP los cuales fueron replanificados para completar en este sprint, como también se puede observar, los mismos

también fueron cumplidos.

Analizando lo sucedido, vemos que el día 25/04 y 28/4 el equipo no avanzó en lo que es el desarrollo del sistema debido a uno de los riesgos mencionados en la tabla de riesgos. Dichos días, los integrantes del equipo tuvieron que dedicar tiempo de estudio a otras asignaturas que se encuentran cursando la universidad, por lo cual no se pudo seguir con el desarrollo habitual del sistema. De tal manera que esto impactó en que el sistema se haya quedado 'estancado' durante dichos días.

Un punto muy importante que se puede notar, es el tiempo de dedicación el cual se le destinó a la creación de interfaces de usuario. A las mismas, el equipo las cuestiono como que no afectarían en gran medida el desarrollo, poniendo en la estimación un mayor énfasis a la hora del desarrollo de la lógica para realizar la conexión entre el BackEnd y el FrontEnd, esto se pudo visualizar en la ejecución del sprint 1. Pero durante la ejecución del sprint 2, el equipo tuvo en consideración lo mencionado. Siendo así que una vez comenzada la ejecución del sprint 2, se tomó en consideración la proporción de tiempo que conlleva realizar interfaces de calidad. Como resultado final del sprint, las interfaces de usuario mantuvieron la calidad con las cuales fueron comenzadas desde el inicio del proyecto.

Siendo lo dicho anteriormente, que al poner énfasis en la calidad de las interfaces, se notó menos tiempo de dedicación para el desarrollo de las interfaces, dado que lo aprendido durante la ejecución del sprint 1. Por otro lado, la correspondiente dedicación que conlleva aplicar patrones y calidad en el código, implica mayor tiempo aún, por lo cual dado lo aprendido del sprint anterior y tomando las medidas correctivas descritas en el mismo, se notó que durante la ejecución del sprint que la división entre el desarrollo de código de calidad y interfaces de usuario fue realizado de forma equitativa. Es por esto último, que dado el balance logrado, se pudo completar las tareas propuestas. Concluyendo así, que el equipo al haber notado un desenlace en el sprint anterior, tomó las medidas correspondientes para lograr los resultados propuestos.

## Burndown chart Sprint 2

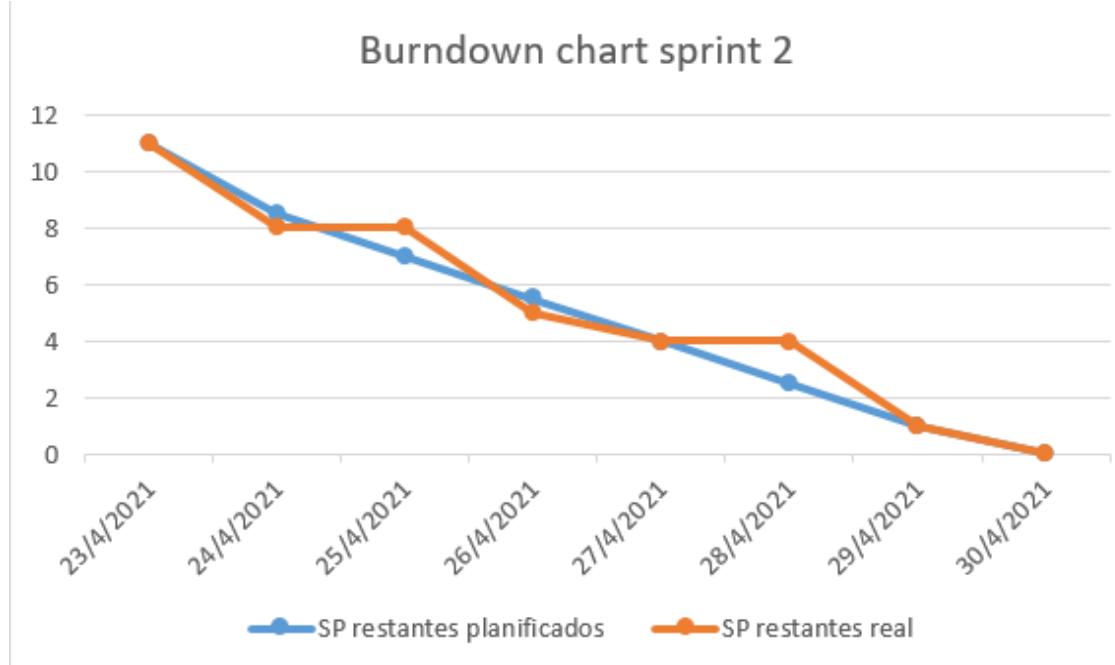


Figura 2.6: Burndown chart del Sprint 2

Fecha	SP restantes planificados	SP restantes real
23/4/2021	11	11
24/4/2021	8,5	8
25/4/2021	7	8
26/4/2021	5,5	5
27/4/2021	4	4
28/4/2021	2,5	4
29/4/2021	1	1
30/4/2021	0	0

Figura 2.7: Comparación día a día: SP restantes planificados VS SP restantes real

### 2.5.6. *Sprint retrospective*

Como equipo, podemos notar que como aspecto positivo, la buena comunicación desarrollada entre los miembros del equipo, de acuerdo a las *Daily Scrum* realizadas y otras conversaciones por fuera cuando alguno de los desarrolladores tuvo algún inconveniente de implementación, o tuvo que tomar alguna decisión importante, que

requirió de la opinión del resto de los miembros del equipo.

Otro punto a destacar como positivo es que al igual que en el sprint anterior, se mantuvieron los estándares de calidad y cobertura de prueba acordados, así como también la revisión de código de a pares, mediante la solicitud de *pull request* al finalizar cada funcionalidad acordada.

Como punto negativo, notamos que el *GitHub Board* en donde están alojadas las tareas a desarrollar, se encuentra un tanto desactualizado con respecto a la realidad. Esto ocurrió básicamente porque al tener un contacto bastante fluido entre los miembros del equipo, algunos miembros ya tenían bien en claro cuales eran sus tareas a desarrollar y olvidaron actualizar el tablero. Este es un aspecto a mejorar, para que todos tengamos un entendimiento claro sobre el punto en que estamos parados del proyecto, y evitar tener que estar consultando constantemente como van las tareas a otros miembros del equipo.

Como conclusión, para el siguiente sprint debemos seguir manteniendo el ritmo de trabajo que venimos llevando adelante, e incluso incorporar para el siguiente sprint algunas de las tareas que estaban planificadas para el sprint 4, debido a que consideramos que las tareas del siguiente sprint (login y logout) fueron sobre-estimadas en la planificación.

Por último, como se mencionó anteriormente, es necesario hacer un esfuerzo mayor para tener el *GitHub Board* lo más actualizado posible (incentivando a que cuando un miembro ve que otro no ha actualizado se lo haga saber por alguna de las vías de comunicación utilizadas), con el fin de que todos sepamos en qué estado se encuentran las tareas en todo momento.

## 2.5.7. Sprint 3 (Original)

### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

### Objetivo del sprint

Implementar de forma completa la funcionalidades de login y logout, que permitirán a los usuarios autenticarse y salir del sistema respectivamente.

### Historias de usuario

- 1.2.4 Login.
- 1.2.5 Logout.

### División de historias de usuario en tareas

Sprint 3	Historias de usuario	Tareas	Estimación	Estimación total
1.2.4 Login	Construcción del esqueleto de la interfaz dentro de la aplicación. Siendo este un formulario para ingresar nombre de usuario y password.	3 hs persona	23 hs persona	
	Lógica para realizar el login	15 hs persona		
1.2.5 Logout	Construcción del esqueleto de la interfaz dentro de la aplicación. Siendo este un botón para salir de la sesión iniciada.	1 hs persona	23 hs persona	
	Lógica para realizar el logout	4 hs persona		

Figura 2.8: División de historias del sprint en tareas, y estimación de las mismas.

### Periodo de fechas

Este sprint tiene como fecha de inicio el 31/04/2021 y como fecha de finalización el 07/05/2021.

## 2.5.8. Sprint 3 (Revisado)

### Velocidad a alcanzar

La velocidad a alcanzar es de 11 SP. Se volvió a estimar el esfuerzo para las tareas de este sprint y se observó que se podría agregar historias de usuario del sprint siguiente.

### Objetivo del sprint

Implementar de forma completa la funcionalidades de login, logout, búsqueda mediante texto y ordenar búsqueda por criterios.

### Historias de usuario

- 1.2.4 Login.
- 1.2.5 Logout.
- 1.2.6 Búsqueda mediante barra de navegación.
- 1.2.8 Ordenar búsqueda por criterios.

### División de historias de usuario en tareas

	<b>Historias de usuario</b>	<b>Tareas</b>	<b>Estimación</b>	<b>Estimación total</b>
<b>Sprint 3</b>	1.2.4 Login	Construcción del esqueleto de la interfaz dentro de la aplicación. Siendo este un formulario para ingresar nombre de usuario y password.	3 hs persona	<b>24 hs persona</b>
		Lógica para realizar el login	6 hs persona	
	1.2.5 Logout	Construcción del esqueleto de la interfaz dentro de la aplicación. Siendo este un botón para salir de la sesión iniciada.	1 hs persona	
		Lógica para realizar el logout	2 hs persona	
	1.2.6 Búsqueda mediante barra de navegación	Spike para investigar cómo mostrar dos columnas en el RecyclerView	2 hs persona	
		Construcción del esqueleto, compuesto por la barra de búsqueda y los productos resultantes de la búsqueda	1 hs persona	
		Lógica de traer productos por texto introducido	2 hs persona	
	1.2.8 Ordenar búsqueda por criterios	Lógica de algoritmos de ordenación y traer los productos ordenados por el criterio seleccionado	7 hs persona	

Figura 2.9: División de historias del sprint en tareas, y estimación de las mismas.

### Periodo de fechas

Este sprint tiene como fecha de inicio el 31/04/2021 y como fecha de finalización el 07/05/2021.

### 2.5.9. Sprint review

Como se puede apreciar en la siguiente *Burndown Chart*, el equipo cumplió con notoriedad todo lo planificado para el sprint 3. En particular además de cumplir con lo planificado, sucedió que dado el avance en conocimiento de las tecnologías se pudo finalizar con anterioridad el sprint (1 día). Cabe destacar que en un principio este sprint contenía únicamente las historias de usuario de *Login* y *Logout*, los cuales viendo el avance obtenido en el sprint anterior (sprint 2), se re-estimaron las duraciones de las tareas, permitiendo que para este sprint se sumen a las historias de usuario mencionadas, la búsqueda mediante barra de navegación y ordenar la búsqueda por criterios.

Analizando lo sucedido, vemos que entre los días 2/5 y 3/5 el equipo se mantuvo trabajando con un ritmo desacelerado debido a que dos de los tres miembros del mismo, el día 3/5 tuvieron una evaluación correspondiente a una asignatura de la Universidad. Dicho evento se encontraba ya categorizado dentro de los riesgos mencionados en la tabla. Por lo cual el ritmo fue desacelerado debido a que dichos integrantes tuvieron que dedicar días de estudio para dicha evaluación, siendo así que esto propagó que durante dichos días estuviera trabajando únicamente uno de los miembros, afectando así el desarrollo habitual del sistema.

#### Burndown chart Sprint 3

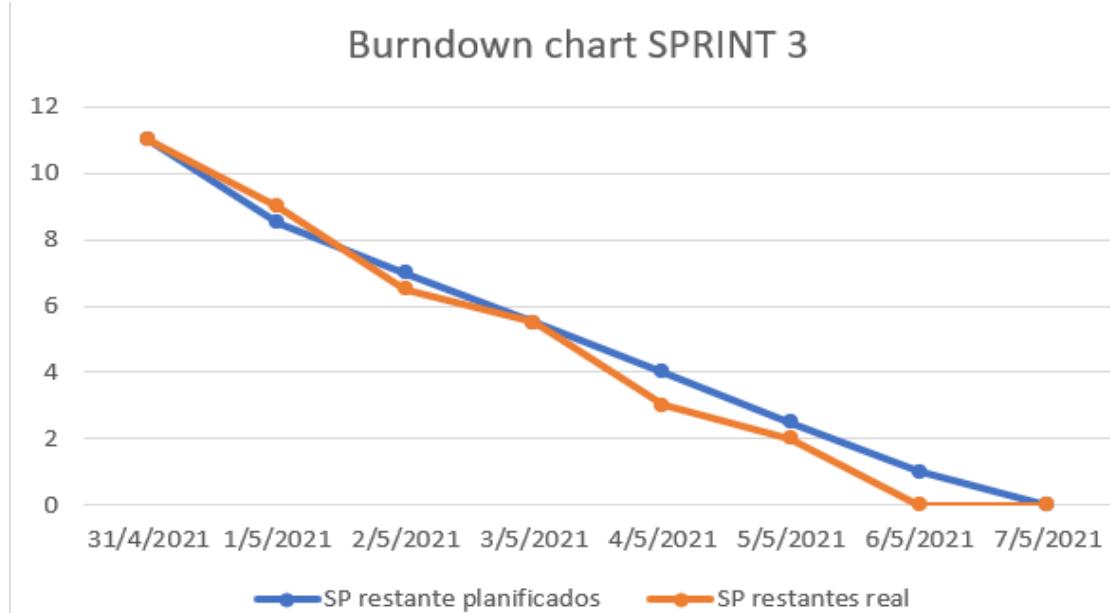


Figura 2.10: Burndown chart del Sprint 3

Fecha	SP restante planificados	SP restantes real
31/4/2021	11	11
1/5/2021	8,5	9
2/5/2021	7	6,5
3/5/2021	5,5	5,5
4/5/2021	4	3
5/5/2021	2,5	2
6/5/2021	1	0
7/5/2021	0	0

Figura 2.11: Comparación día a día: SP restantes planificados VS SP restantes real

Donde un punto muy importante a mencionar dentro de la ejecución fue el manejo de los issues. Tal como se mencionó en la sección correspondiente, los issues que fuesen detectado se reportaría mediante la herramienta GitHub, tal como se puede observar en la siguiente imagen.

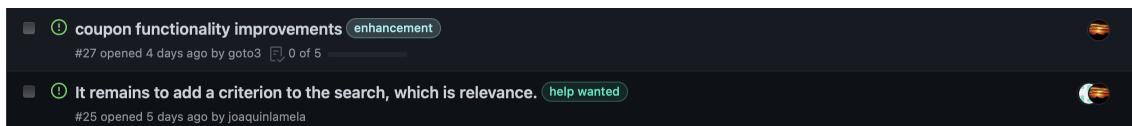


Figura 2.12: Issues reportados

Nuevamente recalando también lo expresado en sprint anteriores, se mantuvo el énfasis sobre la calidad de las interfaces, haciendo que las mismas sean amigables y de calidad. Esto debido a que la interfaz de usuario o user interface (UI) engloba los recursos, sistemas y metodologías que facilitan la interacción de las personas. El objetivo de su desarrollo es que el usuario puede comunicarse con la aplicación de la forma más sencilla e intuitiva posible. Esto nos hace pensar en la frase de Steve Jobs la cual decía 'El diseño no es solo la apariencia, el diseño es cómo funciona', concluyendo así que las características que debería tener una interfaz de usuario, son las siguientes:

- Simplicidad: Cuantos menos elementos decorativos y/u ornamentales formen parte de una interfaz, más fácil será la identificación de los elementos funcionales. **Podemos notar que claramente las diferentes funcionalidades que tiene la aplicación cumple con esta característica, al punto de que no se llena de elementos decorativos que sobrecarguen la pantalla que se esta observando. Esto también respetando los diseños del prototipado realizado.**
- Claridad: Es recomendable nombrar los elementos del menú y cualquier otro tipo de elemento de acción por su nombre y con claridad. Con esto le facilitaremos la navegación al usuario. **Nuevamente podemos notar como este punto es cumplido por la aplicación que se esta desarrollando,**

donde claramente cada elemento gráfico que requiera una acción se encuentra plenamente identificado.

- Familiaridad: Aprovechando que la mayoría de usuarios navegan por sitios web y aplicaciones móvil a diario, hay ciertos elementos de una interfaz que resultan familiares a casi todos los usuarios. El uso de iconos universales como el "menú hamburguesa." en apps, entre otros iconos, nos ayudarán a crear una relación de familiaridad entre la interfaz y el usuario, que le facilitará la navegación y mejorará su experiencia de usuario. **Este punto también es cumplido por nuestra aplicación, al punto del que menú utiliza el icono "menú hamburguesa", entre otros.**
- Rapidez: Cuando nos referimos a que una Interfaz de Usuario sea rápida, lo hacemos en el sentido del aprendizaje. Que un usuario se familiarice y conozca como navegar por una Interfaz de Usuario de forma rápida, es esencial y muy importante para que cada vez que el usuario visite nuestro aplicación no tenga que aprender desde 0. **Este último punto esta muy relacionado con los últimos tres puntos, en el sentido de que si la aplicación es simplista, clara y es universal permitirá que el usuario aprenda con rapidez a utilizarla. Lo cual hace que nuestra aplicación cumpla ampliamente esta característica.**

#### 2.5.10. *Sprint retrospective*

Como equipo, podemos notar que como aspecto positivo, la buena comunicación desarrollada entre los miembros del equipo, de acuerdo a las *Daily Scrum* realizadas. Como forma de mejorar aún más la comunicación entre los miembros y que la misma sea mas fluida, se creó un chat en la herramienta *Discord* con los 3 miembros del equipo, con el fin de intercambiar dudas, inquietudes y sugerencias en relación al proyecto.

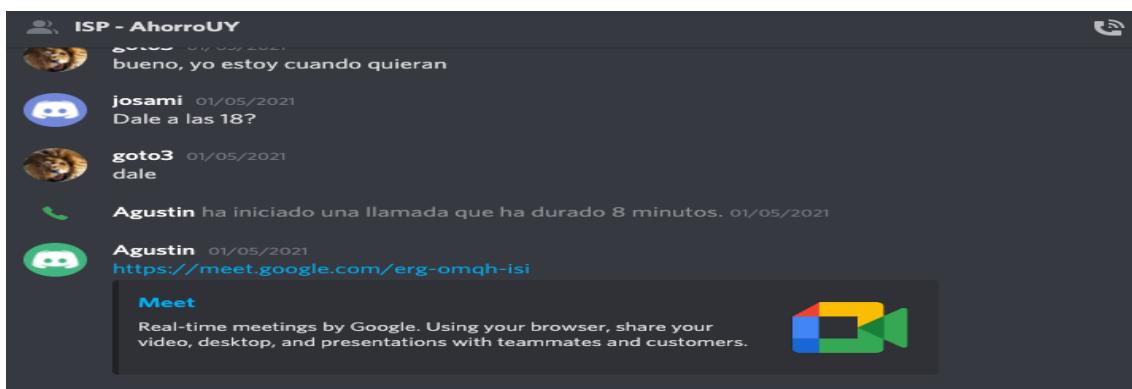


Figura 2.13: Grupo de discord: ISP - AhorroUY

Como punto positivo, se mejoró con respecto al *sprint* anterior el manejo del *Github Board*, pudiendo de esta forma, todos los miembros del equipo conocer en todo momento el estado del proyecto, tal como se puede apreciar en la siguiente figura.

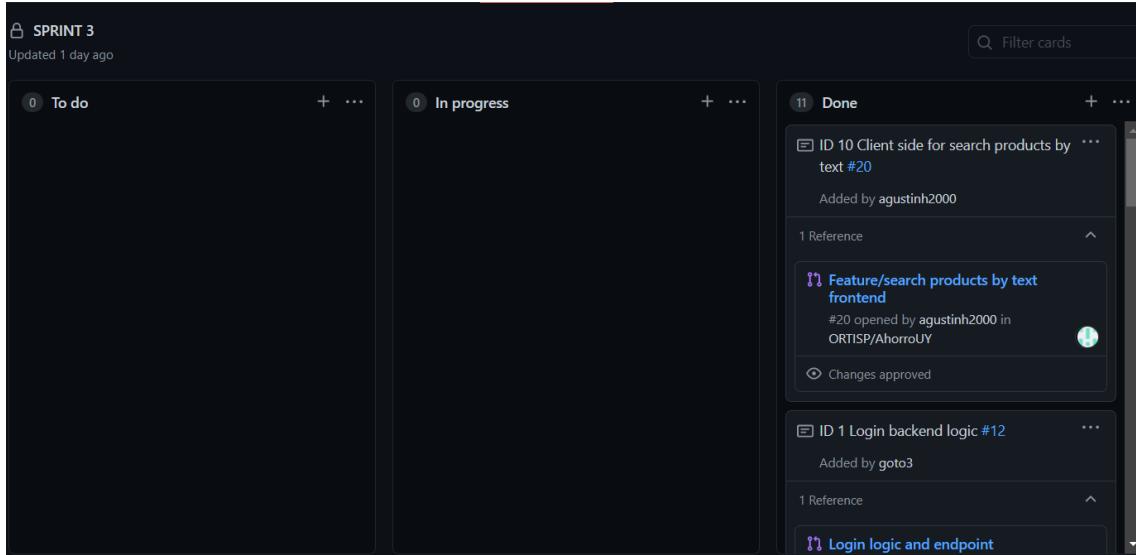


Figura 2.14: *Github board* para el sprint 3.

Como conclusión, para el siguiente sprint debemos seguir manteniendo el ritmo de trabajo que venimos llevando adelante, e incluso incorporar para el siguiente sprint algunas de las tareas que estaban planificadas para el sprint 5. En particular, considerando las re-estimaciones, se incluirá en el siguiente *sprint* la historia de usuario: '1.2.9 Filtrado de productos según categoría.', que inicialmente estaba planificada para el *sprint* 5. La replanificación de dicho *sprint* en virtud del conocimiento adquirido de las tecnologías a utilizar, se puede apreciar con mayor detalle en la siguiente sección.

## 2.5.11. Sprint 4 (Original)

### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

### Objetivo del sprint

Implementar de forma completa la funcionalidades de: búsqueda mediante barra de navegación, búsqueda mediante código de barras, así como también, ordenar las búsquedas por criterios (relevancia, menor precio, mayor precio y alfabético).

### Historias de usuario

- 1.2.6 Búsqueda mediante barra de navegación.
- 1.2.7 Búsqueda mediante código de barras.
- 1.2.8 Ordenar búsqueda por criterios.

### División de historias de usuario en tareas

	<b>Historias de usuario</b>	<b>Tareas</b>	<b>Estimación</b>	<b>Estimación total</b>
<b>Sprint 4</b>	1.2.6 Búsqueda mediante barra de navegación.	Construcción del esqueleto, compuesto por la barra de búsqueda y los productos resultantes de la búsqueda.	2 hs persona	<b>20 hs persona</b>
		Lógica de traer productos por el texto introducido.	5 hs persona	
	1.2.7 Búsqueda mediante código de barras.	Búsqueda de librería de escaneo de códigos de barras	1 hs persona	
		Construcción del esqueleto, compuesto por el botón de escanear.	1 hs persona	
	1.2.8 Ordenar búsqueda por criterios.	Lógica de traer productos por el código de barra.	3 hs persona	
		Construcción del esqueleto, compuesto por un combo con los criterios de ordenación	1 hs persona	
		Lógica de algoritmos de ordenación, y traer los productos ordenados por el criterio seleccionado.	7 hs persona	

Figura 2.15: División de historias del sprint en tareas, y estimación de las mismas.

## Periodo de fechas

Este sprint tiene como fecha de inicio el 10/05/2021 y como fecha de finalización el 17/05/2021.

### 2.5.12. Sprint 4 (Revisado)

#### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

#### Objetivo del sprint

Implementar de forma completa la funcionalidades de: búsqueda mediante barra de navegación, búsqueda mediante código de barras, ordenar las búsquedas por criterios (relevancia, menor precio, mayor precio y alfabético), así como también, filtrar la búsqueda de productos por una determinada categoría.

#### Historias de usuario

- 1.2.7 Búsqueda mediante código de barras.
- 1.2.8 Ordenar búsqueda por criterios.
- 1.2.9 Filtrado de productos según categoría.

#### División de historias de usuario en tareas

	Historias de usuario	Tareas	Estimación	Estimación total
Sprint 4	1.2.7 Búsqueda mediante código de barras	Búsqueda de librería de escaneo de código de barras.	2 hs persona	20 hs persona
		Construcción del esqueleto, compuesto por el botón de escanear.	2 hs persona	
		Lógica de traer productos por el código de barra.	5 hs persona	
	1.2.8 Ordenar búsqueda por criterios	Construcción del esqueleto, compuesto por un combo con los criterios de ordenación.	1 hs persona	
		Construcción del esqueleto de la interfaz dentro de la aplicación. Siendo este la muestra del texto e imagen asociado a la categoría.	3 hs persona	
		Logica de obtención de las categorías con su correspondiente imagen.	2 hs persona	
	1.2.9 Filtrado de productos según categoría	Logica de obtención de los productos correspondientes a la categoría seleccionada.	4 hs persona	
		Construcción del esqueleto de los productos filtrados por categoría.	1 hs persona	

Figura 2.16: Replanificación. División de historias del sprint en tareas, y estimación de las mismas.

## Periodo de fechas

Este sprint tiene como fecha de inicio el 10/05/2021 y como fecha de finalización el 17/05/2021.

## Sprint review

Como se puede apreciar en la siguiente *Burndown Chart*, el equipo cumplió con notoriedad todo lo planificado para el sprint 4. Cabe destacar que en un principio este sprint contenía únicamente las historias de usuario de 'Búsqueda mediante barra de navegación', 'Búsqueda mediante código de barras' y 'Ordenar búsqueda por criterios', los cuales viendo el avance obtenido en el sprint anterior (sprint 3), se re-estimaron las duraciones de las tareas. Cabe destacar que una de dichas historias de usuario fueron ejecutadas durante la duración del sprint 3 permitiendo así que para este sprint se sumen a las historias de usuario 'Búsqueda mediante código de barras' y 'Ordenar búsqueda por criterio', el filtrado de productos según categoría.

Analizando lo sucedido, vemos que entre los días 12/5 y 13/5 el equipo se mantuvo constante con respecto al desarrollo, debido a que dos de los tres miembros del equipo, el día 13/5 tuvieron una evaluación correspondiente a una asignatura de la Universidad. Dicho evento se encontraba ya categorizado dentro de los riesgos mencionados en la tabla. Por lo cual el ritmo fue desacelerado debido a que dichos integrantes tuvieron que dedicar días de estudio para dicha evaluación. Ocurriendo también que el otro miembro durante dichos días se encontrase enfermo, donde dicho evento también se encuentra contenido dentro de los riesgos mencionado en la tabla. Siendo así que dichos eventos propagaron que durante dichos días el equipo estuviera desarrollando por debajo de lo habitual, afectando así el desarrollo.

## Burndown chart Sprint 4

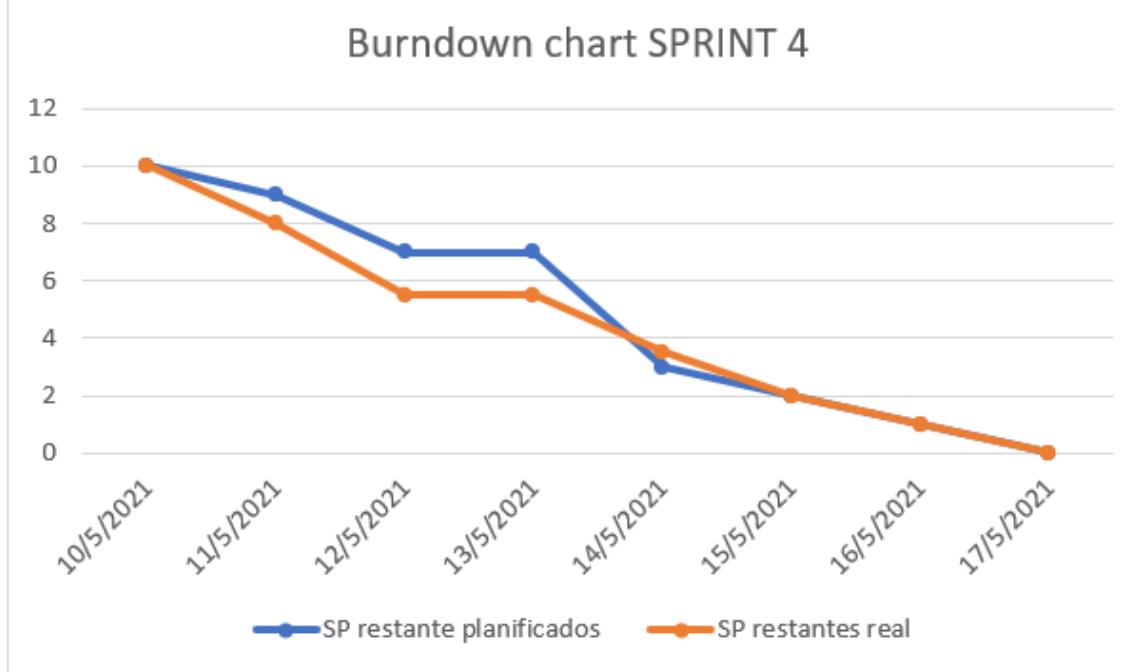


Figura 2.17: Burndown chart del Sprint 4

Fecha	SP restante planificados	SP restantes real
10/5/2021	10	10
11/5/2021	9	8
12/5/2021	7	5,5
13/5/2021	7	5,5
14/5/2021	3	3,5
15/5/2021	2	2
16/5/2021	1	1
17/5/2021	0	0

Figura 2.18: Comparación día a día: SP restantes planificados VS SP restantes real

### Sprint retrospective

Como retrospectiva, podemos decir que hasta el momento el proyecto se viene desarrollando con normalidad, y la metodología *Scrum* se ha aplicado de manera correcta.

Como aspecto positivo, los procesos se han desarrollado de manera correcta, la comunicación entre los miembros del equipo han sido fluidas, el desarrollo de las funcionalidades se han hecho según los estándares de calidad previamente establecido y se ha mantenido el *Github Board* actualizado para que los miembros del equipo puedan saber el estado del proyecto en todo momento.

Entre los miembros del equipo, notamos como punto a realizar en el próximo *sprint*, un análisis estático de código, con las herramientas previamente establecidas, que nos permita identificar con cierta antelación los *bugs* o vulnerabilidades presentes en el código, y de esa forma, poder solucionarlo previo a la liberación final.

### 2.5.13. Sprint 5 (Original)

#### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

#### Objetivo del sprint

Implementar de forma completa la funcionalidades de: poder filtrar la búsqueda de productos según determinadas categorías, poder agregar y visualizar productos favoritos, como también, la funcionalidad que permite obtener al usuario cupones de descuento a utilizar en los supermercados.

## Historias de usuario

- 1.2.9 Filtrado de productos según categoría.
- 1.2.10 Productos favoritos.
- 1.2.11 Cupones de descuento en compras.

## División de historias de usuario en tareas

	<b>Historias de usuario</b>	<b>Tareas</b>	<b>Estimación</b>	<b>Estimación total</b>
<b>Sprint 5</b>	1.2.9 Filtrado de productos según categoría	Construcción del esqueleto de la interfaz dentro de la aplicación. Siendo este la muestra del texto e imagen asociado a la categoría.	2 hs persona	<b>22 hs persona</b>
		Lógica de obtención de las categorías con su correspondiente imagen.	2 hs persona	
		Lógica de obtención de los productos correspondientes a la categoría seleccionada.	4 hs persona	
		Construcción del esqueleto de los productos filtrados por categoría.	1 hs persona	
	1.2.10 Productos favoritos	Construcción del esqueleto, compuesto por la sección "mis favoritos" que contiene la lista de productos favoritos del usuario.	2 hs persona	
		Lógica para obtener productos favoritos.	2 hs persona	
		Lógica de agregar/quitar de la lista de favoritos.	3 hs persona	
	1.2.11 Cupones de descuento	Construcción del esqueleto, compuesta por la sección "Mis cupones" en donde se muestran los QRs que el usuario puede utilizar.	2 hs persona	
		Lógica de generación del código QR y asociación con el usuario que está comprando.	3 hs persona	
		Lógica de obtención de los cupones pertenecientes a un usuario.	1 hs persona	

Figura 2.19: División de historias del sprint en tareas, y estimación de las mismas.

## Periodo de fechas

Este sprint tiene como fecha de inicio el 18/05/2021 y como fecha de finalización el 25/05/2021.

## 2.5.14. Sprint 5 (Revisado)

### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

## Objetivo del sprint

Implementar de forma completa la funcionalidades de: poder agregar y visualizar productos favoritos, como también, la funcionalidad que permite obtener al usuario cupones de descuento a utilizar en los supermercados.

## Historias de usuario

- 1.2.10 Productos favoritos.
- 1.2.11 Cupones de descuento en compras.

## División de historias de usuario en tareas

	Historias de usuario	Tareas	Estimación	Estimación total
Sprint 5	1.2.10 Productos favoritos	Construcción del esqueleto, compuesto por la sección "Mis favoritos" que contiene la lista de productos favoritos del usuario.	6 hs persona	27 hs persona
		Logica para obtener productos favoritos.	5 hs persona	
		Logica de agregar/quitar de la lista de favoritos	8 hs persona	
	1.2.11 Cupones de descuento	Construcción del esqueleto, compuesto por la sección "Mis Cupones" en donde se muestran los cupones con una forma de ver el código QR asociado a cada uno.	4 hs persona	
		Logica de generación del código QR para el usuario	2 hs persona	
		Logica de obtención de los cupones pertenecientes a un usuario	2 hs persona	

Figura 2.20: División de historias del sprint en tareas, y estimación de las mismas.

## Periodo de fechas

Este sprint tiene como fecha de inicio el 18/05/2021 y como fecha de finalización el 25/05/2021.

## Sprint Review

Como se puede apreciar en la siguiente *Burndown Chart*, el equipo cumplió con notoriedad todo lo planificado para el sprint 5. Cabe destacar que en un principio este sprint contenía únicamente las historias de usuario de 'Filtrado de productos

según categorías', 'Productos favoritos' y 'Cupones de descuento', los cuales viendo el avance obtenido en el sprint anterior, se re-estimaron las duraciones de las tareas. Cabe destacar que una de dichas historias de usuario fueron ejecutadas durante la duración del sprint 4, y por lo tanto, en este sprint únicamente se ejecutaron las historias de usuario de: 'Productos favoritos' y 'Cupones de descuento en compras', lo que implico que el equipo pueda avanzar con lo que es la documentación de la aplicación, ya que como se puede observar en la *Burndown chart*, el equipo finalizo dos días antes de lo planificado.

### Burndown chart Sprint 5

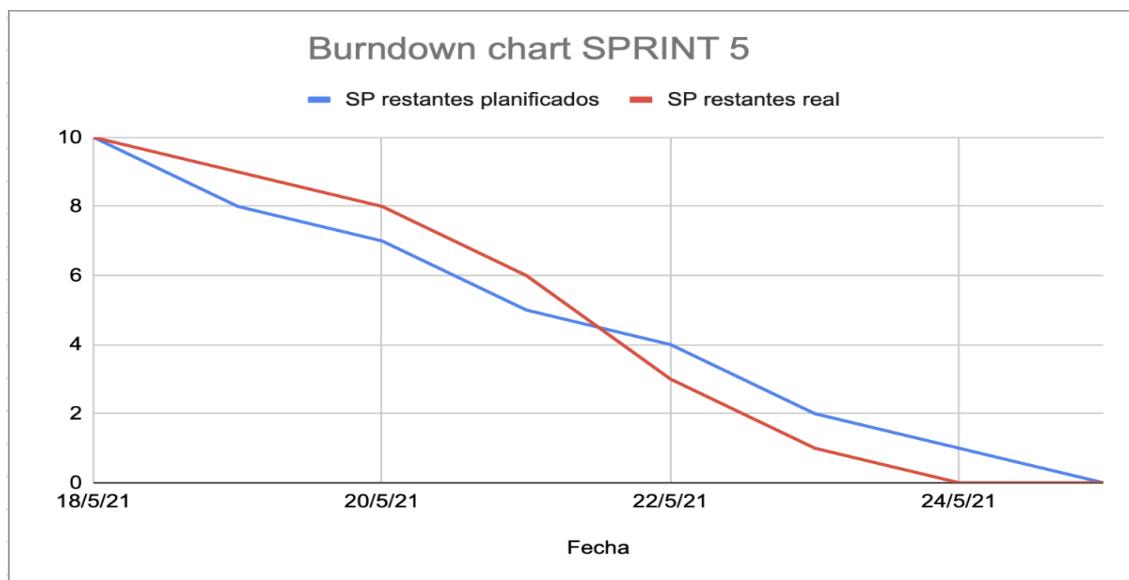


Figura 2.21: Burndown chart del Sprint 5

Fecha	SP restantes planificados	SP restantes real
18/5/21	10	10
19/5/21	8	9
20/5/21	7	8
21/5/21	5	6
22/5/21	4	3
23/5/21	2	1
24/5/21	1	0
25/5/21	0	0

Figura 2.22: Comparación día a día: SP restantes planificados VS SP restantes real

## Sprint Retrospective

Como retrospectiva, podemos decir que hasta el momento el proyecto se viene desarrollando con normalidad, y la metodología *Scrum* se ha aplicado de manera correcta.

Como aspecto positivo, los procesos se han desarrollado de manera correcta, la comunicación entre los miembros del equipo han sido fluidas, el desarrollo de las funcionalidades se han hecho según los estándares de calidad previamente establecido y se ha mantenido el *Github Board* actualizado para que los miembros del equipo puedan saber el estado del proyecto en todo momento.

### 2.5.15. Sprint 6 (Original)

#### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

#### Objetivo del sprint

Implementar de forma completa la funcionalidades de poder visualizar el carrito de compras, así como también, la que permite para un conjunto de productos, obtener el supermercado que ofrece el precio mas conveniente para dicho conjunto de productos.

#### Historias de usuario

- 1.2.12 Visualizar carrito de compras.
- 1.2.13 Obtener la mejor opción de compra para un conjunto de productos.

#### División de historias de usuario en tareas

#### Periodo de fechas

Este sprint tiene como fecha de inicio el 26/05/2021 y como fecha de finalización el 02/06/2021.

	<b>Historias de</b>	<b>Tareas</b>	<b>Estimación</b>	<b>Estimación total</b>
<b>Sprint 6</b>	1.2.12 Carrito de compras	Construcción del esqueleto, compuesto por la sección "carrito de compras" que contiene la lista de productos a comprar, con posibilidad de agregar más cantidad	4 hs persona	
		Lógica de almacenar los productos del carrito en el "cache" local de la aplicación, agregado, eliminación de ellos y visualización de los productos.	5 hs persona	
		Lógica de obtener los productos con su respectiva imagen	2 hs persona	
		Visualizar el carrito de compras en la parte inferior durante búsqueda o pantalla de inicio (hasta x cantidad de productos, luego muestra únicamente la cantidad).	1 hs persona	<b>24 hs persona</b>
	1.2.13 Obtener la mejor opción de compra para un conjunto de productos.	Construcción del esqueleto, compuesta por el lugar donde va el mapa, una sección con los supermercados disponibles y el precio para cada uno de ellos, y un botón que diga	3 hs persona	
		Desarrollo del mapa, colocación de marcadores de supermercados.	4 hs persona	
		Lógica para obtención de los datos de los supermercados que contienen los productos seleccionados y el precio total para esos productos.	4 hs persona	
		Sumar el costo de la compra al total del dinero gastado en compras del usuario, para posteriormente poder generar cupones.	1 hs persona	

Figura 2.23: División de historias del sprint en tareas, y estimación de las mismas.

### 2.5.16. Sprint 6 (Revisado)

#### Velocidad a alcanzar

La velocidad a alcanzar es de 10 SP.

#### Objetivo del sprint

Implementar de forma completa la funcionalidades de poder visualizar el carrito de compras, así como también, la que permite para un conjunto de productos, obtener el supermercado que ofrece el precio mas conveniente para dicho conjunto de productos.

## Historias de usuario

- 1.2.12 Visualizar carrito de compras.
- 1.2.13 Obtener la mejor opción de compra para un conjunto de productos.

## División de historias de usuario en tareas

	Historias de usuario	Tareas	Estimación	Estimación total
Sprint 6	1.2.12 Carrito de compras	Construcción del esqueleto, compuesto por la sección "carrito de compras" que contiene la lista de productos a comprar, con posibilidad de modificar la cantidad.	3 hs persona	52 hs persona
		Logica de almacenar los productos del carrito en el "cache" local de la aplicación, agregado, eliminación de ellos y visualización de los productos	5 hs persona	
		Lógica para integrar la funcionalidad del carrito a funcionalidades existentes como puede ser la pantalla de home, búsqueda y favoritos.	15 hs persona	
		Visualizar el carrito de compras en la parte inferior de la aplicación cuando no está vacío. Incluir un botón para "Buscar supermercados".	12 hs persona	
	1.2.13 Obtener la mejor opción de compra para un conjunto de productos	Construcción del esqueleto, compuesto por el lugar donde va el mapa, una sección con los supermercados disponibles y el precio para cada uno de ellos, junto a un botón de buscar supermercados.	2 hs persona	
		Desarrollo del mapa, colocación de marcadores de supermercados	8 hs persona	
		Logica para la obtención de los datos de los supermercados que contienen los productos seleccionados y el precio total para esos productos.	6 hs persona	
		Sumar el costo de la compra total de dinero gastado en compras del usuario, para posteriormente poder generar cupones.	1 hs persona	

Figura 2.24: División de historias del sprint en tareas, y estimación de las mismas.

## Periodo de fechas

Este sprint tiene como fecha de inicio el 26/05/2021 y como fecha de finalización el 02/06/2021.

## Sprint Review

Como se puede apreciar en la siguiente *Burndown Chart*, el equipo cumplió con todo lo planificado en el sprint 6 tal como se puede observar en la *burndown chart*

que se presenta a continuación.

Por un lado la funcionalidad del carrito llevó mucho mas tiempo que la estimada debido a que se tuvo que adaptar las funcionalidades actualmente implementadas como fueron las del home, búsqueda, y favoritos para que puedan agregar productos a favoritos, para esto se tuvo que reimplementar todos los adaptadores y viewmodels de las funcionalidades mencionadas, además la funcionalidad de mostrar el carrito de compras en la parte inferior llevó mucho mas tiempo del estimado debido a que realizaron varias implementaciones de forma iterativa con la finalidad de obtener una buena calidad en cuanto a experiencia de usuario. Por otro lado la funcionalidad de 'Obtener la mejor opción de compra para un conjunto de productos' insumió mas tiempo del planificado. A pesar de las dificultades experimentadas se pudo llegar a lo planeado para el sprint presente.

### Burndown chart Sprint 6

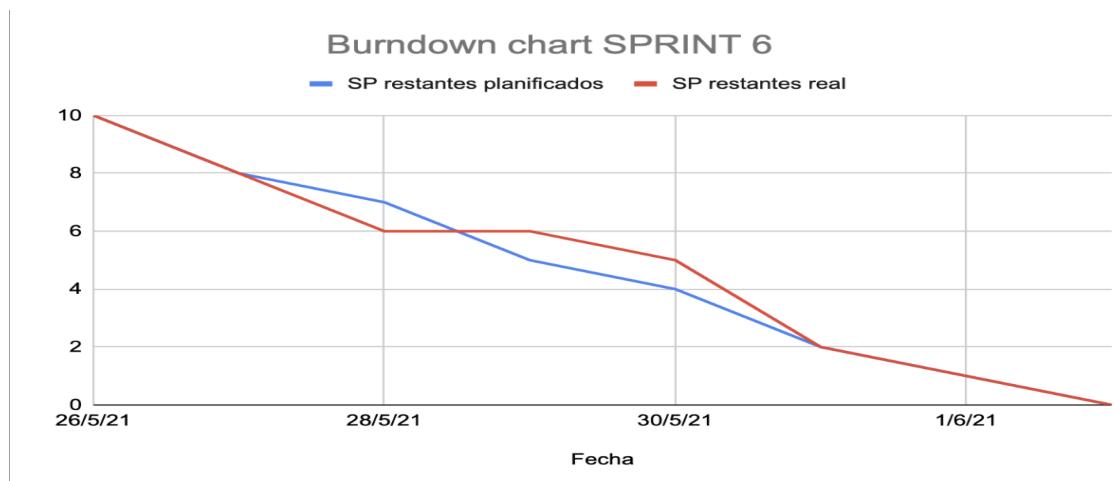


Figura 2.25: Burndown chart del Sprint 6

Fecha	SP restantes planificados	SP restantes real
26/5/21	10	10
27/5/21	8	8
28/5/21	7	6
29/5/21	5	6
30/5/21	4	5
31/5/21	2	2
1/6/21	1	1
2/6/21	0	0

Figura 2.26: Comparación día a día: SP restantes planificados VS SP restantes real

## Sprint Retrospective

Como retrospectiva, siendo el último *sprint* desarrollado por el equipo concluimos que el proyecto se desarrollo con total normalidad y siguiendo la metodología *Scrum* de forma adecuada.

Como un punto a destacar, es que los procesos se desarrollaron de manera correcta, existió en todo momento una buena comunicación entre los miembros del equipo y las funcionalidades se desarrollaron siguiendo los estándares que previamente fueron acordados.

Si bien no hay ningún otro sprint fijado debido a que el equipo ya finalizó con todo lo que refiere a los aspectos funcionales de la aplicación, debemos seguir trabajando en lo que refiere a la documentación que deje lo suficientemente claro la arquitectura, decisiones de diseño, especificación de la API, entre otros, así como también, ajustar algunos pequeños detalles de lo que es la UI de la aplicación. Estas mejoras fueron agregadas como issues en el repositorio del proyecto las cuales se fueron solucionando una vez terminado el sprint.

## 2.6. Gestión de riesgos

En esta sección, se hablará acerca de la definición de riesgos y el registro de los posibles riesgos que podrían suceder durante la ocurrencia del proyecto. Es por esto que debemos primero hablar de riesgo. El riesgo de un proyecto es un evento o condición incierta que, de producirse, tiene un efecto positivo o negativo en uno o más de los objetivos del proyecto, tales como el alcance, el cronograma, el costo y la calidad.

De manera que, una buena gestión de riesgos tiene como objetivo aumentar la probabilidad y el impacto de los eventos positivos (oportunidades), y disminuir la probabilidad y el impacto de los eventos negativos (amenaza) en el proyecto. Por lo cual se puede determinar que la característica fundamental:

1. Las oportunidades son eventos positivos, no considerados en la planificación inicial, que mejoran el tiempo del proyecto, cambian el alcance y el cronograma.
2. Por el contrario, las amenazas son eventos negativos considerados y no considerados, en la planificación, que pueden empeorar el tiempo del proyecto, el alcance del mismo y el cronograma, dependiendo de los planes de contingencia que tenga el equipo.

Siendo así que para llevar un correcto registro de los riesgos, se siguió el siguiente formato:

1. **Id:** Identificador del riesgo.
2. **Fecha de ingreso:** Fecha de ingreso del riesgo.
3. **Componente:** Componente que se verá afectado por la ocurrencia de dicho riesgo.
4. **Condición:** Condición que debe ocurrir para que se concrete el riesgo.
5. **Consecuencia:** Son las consecuencias que traerá el riesgo si hay una ocurrencia del mismo.
6. **Tipo:** El tipo de los riesgos la categorizaremos en: amenaza o oportunidad.
7. **Probabilidad:** Es la probabilidad de que el riesgo ocurra. La escala de los valores se puede observar en la siguiente tabla:

Valor	Descripción
1	Baja probabilidad de ocurrencia
2	Media probabilidad de ocurrencia
3	Alta probabilidad de ocurrencia

8. **Impacto:** Es el impacto que tiene el riesgo en el proyecto, si el mismo ocurre. En este caso se puede observar la clasificación en la siguiente tabla:

Valor	Descripción
1	Bajo impacto
2	Impacto medio
3	Alto impacto

9. **Exposición:** Es el producto entre la probabilidad de ocurrencia y el impacto de un riesgo.
10. **Acción de respuesta:** Acción de respuesta específica para implementar la estrategia de respuesta adoptada, para prevenir el riesgo o disminuir su impacto.
11. **Acción de contingencia:** Acción a ejecutar en caso de que el riesgo ocurra para lograr minimizar el impacto del mismo.
12. **Disparador:** Evento que dispara la ejecución de la acción de contingencia.
13. **Responsable:** Persona responsable del riesgo.

Id	Fecha de Ingreso	Componente	Riesgo			Prioridad		
			Condición	Consecuencia	Tipo	Probabilidad	Impacto	Exposición
1	03/30/2021	Implantación	Al desarrollar las funcionalidades, surgen bugs críticos que requieren ser solucionados para continuar	Se retrasa el cronograma	Amenaza	2	3	6
2	03/30/2021	Ambiente de desarrollo	Las herramientas de desarrollo no funcionan como se esperaba, el equipo de desarrollo necesita tiempo para resolverlo o adaptarse a las nuevas herramientas	Retraso en el proyecto	Amenaza	1	2	2
3	03/30/2021	Personal	Renuncia de algún integrante del proyecto	Atraso del proyecto según integrante	Amenaza	1	2	2
4	03/30/2021	Proyecto	Los ciclos de revisión/decisión del cliente para los planes, prototipos y especificaciones son más lentos de lo esperado	Que el equipo de desarrollo no pueda comenzar a desarrollar en la fecha estimada y se retrasen todas las tareas siguientes	Amenaza	2	2	4
5	03/30/2021	Implantación	Tiempo de desarrollo de pruebas unitarias mayor al planificado.	Retraso en el tiempo a la hora de probar cada modulo	Amenaza	1	1	1
6	03/30/2021	Proyecto	Se añaden nuevos requisitos que no estaban incluidos en el plan de proyecto.	Que deban implementarse dichos requisitos y que el tiempo estimado para el proyecto no sea suficiente	Amenaza	1	3	3
7	03/30/2021	Ambiente de desarrollo	Que alguna de las librerías indispensables para desarrollar alguna de las funcionalidades no se encuentre disponible.	Buscar alternativas de librerías para satisfacer el requerimiento.	Amenaza	1	2	2
8	03/30/2021	Proyecto	La planificación realizada no incluye tareas necesarias	Retraso en las tareas planificadas y necesidad de actualizar las historias de usuario que entran en cada sprint.	Amenaza	2	3	6
9	03/30/2021	Proyecto	Tiempo insuficiente para realizar controles de calidad	Retraso en los siguientes sprint	Amenaza	2	2	4
10	03/30/2021	Proyecto	La curva de aprendizaje para entender las nuevas tecnologías es más larga de lo esperado.	Mayor tiempo y dificultad en la implementación de las funcionalidades resultando en un retraso de los tiempos de puesta en marcha	Amenaza	2	3	6
11	03/30/2021	Proyecto	Estimación del esfuerzo de desarrollar las funcionalidades sea mayor al estimado	Incrementar el tiempo de desarrollo	Amenaza	2	2	4
12	03/30/2021	Proyecto	Que los integrantes del equipo no puedan dedicar todo el tiempo estipulado al proyecto	Retraso en el desarrollo de las tareas en las que son responsables	Amenaza	1	3	3

Figura 2.27: Riesgos (parte 1/2).

Plan de Respuesta			
Acción de Respuesta	Acción de Contingencia	Disparador	Responsable
Realizar revisiones periodicas de codigo	Realizar solicitud de cambio, analizar impacto y corregir cada bug critico detectado	Al realizar un análisis estático de código	Desarrollador
Verificar de forma temprana las herramientas de desarrollo con que las que se cuenta.	Resolver las herramientas de desarrollo con las que se cuenta o adaptarse a nuevas herramientas.	Implementando una funcionalidad	Desarrollador
Medir el clima laboral de los integrantes del proyecto. Promover actividades de Integración. Hacer un seguimiento personal de los integrantes del Equipo	Integrar otra persona al equipo de proyecto que pueda cumplir el rol de la persona saliente, en forma eficiente.	Seguimiento	-Equipo proyecto
Buscar metodos mas practicos y efectivos de comunicación para desarrollar con el cliente.	Tomar decisiones conforme a lo estipulado en el plan de proyecto y seguir adelante.	Al llegar la fecha estipulada para el ciclo de revisión con el cliente, no se obtiene una respuesta por parte del mismo	- Cliente - Equipo de proyecto
Seguimiento semanal, imponiendo que se realicen pruebas unitarias en paralelo al desarrollo de cada funcionalidad.	De darse el caso se deberá evaluar en conjunto con el cliente si se desea retrasar el cronograma o priorizar el desarrollo de pruebas unitarias	Desvio de tiempo a la hora de codificar pruebas unitarias	- Tester
Mantener una comunicación semanal con el cliente que nos ayude a detectar de forma temprana las nuevas necesidades que tiene	Realizar solicitud de cambio, analizar el impacto, en base a esto ajustar lo planificado en los sprints, y por ultimo implementar la funcionalidad.	Comunicación del cliente	- Cliente
Realizar un analisis previo al comienzo del proyecto, de las librerías a utilizar y los requerimientos de las mismas para que funcionen.	Buscar una librería alternativa que sea compatible	Al intentar implementar una funcionalidad que utilice la librería	- Desarrollador
Buena planificación de las tareas a desarrollar, contemplando todas las tareas necesarias para satisfacer los requerimientos.	Ajustar los sprint para realizar las tareas necesarias.	Al realizar una actividad el equipo toma conocimiento que previo a la misma de debería haber hecho otra que no está planificada.	- Equipo de trabajo
Dejar algunos días entre la finalización del proyecto y la fecha de entrega para realizar actividades de aseguramiento de calidad.	Definir si se va a seguir con los mismos estándares de calidad o si se va a ajustar la revisión para cada sprint asegurandose de la calidad del producto.	Desvio de tiempo al realizar un control de calidad de la funcionalidad introducida.	- Gerente de calidad
Entender el código en forma temprana.	Ajustar los sprint, destinando una mayor cantidad de tiempo a las actividades relacionadas a la comprensión de las nuevas tecnologías.	Desvio de tiempo al inicio el proyecto cuando se busca entender nuevas tecnologías.	- Desarrollador
Orientar la solución a una solución mínima viable, cumpliendo con los requerimientos básicos del cliente	Replanificar el Proyecto	Desvio de tiempo a la hora de realizar una funcionalidad	- Tester
Cumplimiento estricto de los tiempos asignados a las tareas	Ajustar el Plan de Proyecto	Desvio en el tiempo destinado	- Gerente de Proyecto

Figura 2.28: Riesgos (parte 2/2).

### 2.6.1. Minimización de riesgos al usar Scrum

El hecho de elegir Scrum como metodología ágil de trabajo, trae ventajas respecto a la gestión de los riesgos:

- **Flexibilidad:** La flexibilidad reduce los riesgos relacionados con el negocio, debido a que en cualquier momento se pueden introducir o eliminar requisitos, y el cliente está involucrado desde la perspectiva del negocio.
- **Feedback:** El Feedback periódico reduce los riesgos relativos a expectativas: Así, frecuentemente, se puede responder a la pregunta “¿Hay algún requisito que ha sido mal-entendido?».
- **Propiedad del proyecto:** La propiedad del proyecto por parte del equipo de desarrollo y su responsabilidad colectiva respecto al éxito o fracaso del proyecto, reduce los riesgos de estimaciones: al crear un mayor compromiso del equipo en alcanzar sus objetivos.
- **Transparencia:** El tablero de Scrum, en el que se indican las actividades a realizar por el equipo y que todos pueden consultar. En él se incluyen los impedimentos, que el equipo va encontrando y que pueden afectar al resultado del sprint. Es el rol del Scrum Master quien tiene que gestionarlos para evitar que impacten en la eficiencia del equipo.
- **Entrega iterativa:** La entrega iterativa reduce los riesgos de inversión: De forma que al ir disponiendo de productos “usables” muy pronto, se detecta antes las posibles incongruencias con el mercado y así ser capaces de reorientar el producto.

## 2.6.2. Análisis de probabilidad de ocurrencia

En la siguiente tabla que se presenta a continuación, se observa para cada riesgo, el análisis de la probabilidad de ocurrencia de los mismos. Estos análisis fueron basados en la realidad existente y el contexto actual.

Id del riesgo	Justificación
1	Probabilidad media de ocurrencia, ya que consideramos que a pesar de que realicemos revisiones periódicas, la posibilidad de introducir bugs siempre está presente.
2	Baja probabilidad de ocurrencia, ya que previo al proyecto se chequeó el estado de cada una de las herramientas necesarias para la ejecución del proyecto, y es difícil que haya cambios significativos en ese sentido.
3	Baja probabilidad de ocurrencia, ya que a priori se aprecia una buena comunicación y disposición de los miembros del equipo.
4	Probabilidad media de ocurrencia, ya que consideramos que al haber tantos equipos realizando aplicaciones, el cliente recibirá muchos avances y tendrá que dar muchos feedbacks, lo que puede llevar, a que las revisiones por parte del mismo se atrasen un poco.
5	Probabilidad media de ocurrencia, ya que la cantidad de pruebas a introducir después de realizar cada nueva funcionalidad no la sabemos con exactitud, y ese número puede ser superior al planificado
6	Baja probabilidad de ocurrencia, ya que no se ha notado que el cliente tenga la intención de agregar nuevos requisitos por fuera de los planificados.
7	Baja probabilidad de ocurrencia, ya que nos hemos informado y las librerías tiene un soporte robusto y están disponibles para utilizarlas de manera sencilla, pero el riesgo está presente porque no conocemos exactamente las versiones que actualmente requiere la librería.
8	Probabilidad media de ocurrencia, ya que si bien consideramos que realizamos una buena planificación de actividades, la posibilidad de que ocurran actividades imprevistas que no fueron planificadas siempre está presente.
9	Probabilidad media de ocurrencia, ya que consideramos que suele ocurrir que por temas de tiempo, se prioriza el realizar todas las <i>features</i> definidas en el alcance del proyecto, y se pierde un poco el foco en realizar controles de calidad.
10	Probabilidad media de ocurrencia, ya que el equipo no cuenta con conocimiento previo ni experiencia utilizando las tecnologías.
11	Probabilidad media de ocurrencia, ya que en cierto modo está relacionado con el riesgo 10, si en cierto modo este último se materializa, llevará a que este también ocurra
12	Probabilidad baja de ocurrencia, ya que se nota un compromiso importante de los miembros del equipo, no obstante hay una baja probabilidad de que ocurra producto de que en las próximas semanas los miembros del equipo estarán trabajando en paralelo en otros proyectos.

# 3. Gestión de la configuración

## 3.1. Control de cambios y de versiones

### 3.1.1. Estructura del repositorio y liberaciones identificadas

#### GitFlow

El uso de ramas, para administrar y diferenciar el trabajo, y a su vez para identificar distintos ambientes (producción, pruebas, desarrollo) son de las principales utilidades que dispone Git, esto nos ayuda a llevar un mejor control del código y del proyecto. Una rama se trata de una bifurcación del estado del código que crea un nuevo camino de cara a la evolución del código, en paralelo a otras ramas que se puedan generar. En nuestro caso, decidimos hacer uso de Git mediante 2 ramas fijas (master y develop) y otros 3 tipos de ramas: features (funcionalidades), bugfix (bugs) y refactor.

La rama master la utilizamos cada vez que llegamos a una etapa estable que tuviera algún flujo completamente cerrado. Sería como la rama de producción de nuestro proyecto.

La rama develop es donde mergeamos cuando una de las ramas features/bugfix/refactor fue revisada y no tiene errores (por lo menos al probar manualmente el desarrollador). Sería como la rama de pruebas para que el cliente pueda ver la última versión del producto y pueda probar.

Cada tarea en la cual trabajamos a lo largo del obligatorio la podemos catalogar bajo una de las tres tipos de ramas comentadas más arriba. Para cada nuevo trabajo que realizamos, creamos una rama que partiera de develop y cuando el trabajo está finalizado y testeado, subimos el branch y creamos un pull request, por lo general acompañado de una breve descripción y una referencia a la card de GitHub Board correspondiente.

Luego está el proceso de validación del pull request creado, el miembro del equipo que no trabaja en esa funcionalidad se encarga de hacer una revisión del código, y si considera necesario puede dejar comentarios, que le pueden servir de ayuda al desarrollador para realizar cambios en pos de mejoras en cuanto a estándares de codificación, detección de errores, entre otros. En caso de que considere que todo está funcionando correctamente, aprueba el pull request y se hace un merge de la

rama sobre develop. Luego el flujo se reinicia.

En cuanto a las ramas refactor, las creamos con los objetivos de: mejorar la facilidad de comprensión del código y eliminar código muerto. En el proceso de refactorización, no arreglamos errores ni incorporamos funcionalidades, sino que alteramos la estructura interna del código sin cambiar el comportamiento externo.

Las principales ventajas de utilizar este modelo son, entre otras, disminuir la posibilidad de mezclar ramas, independizar partes del código para el momento que entregamos a producción, el equipo de pruebas recibe más rápido código para probar y así evitamos que muchos bugs se "pasen por alto" o que lleguen a producción por accidente.

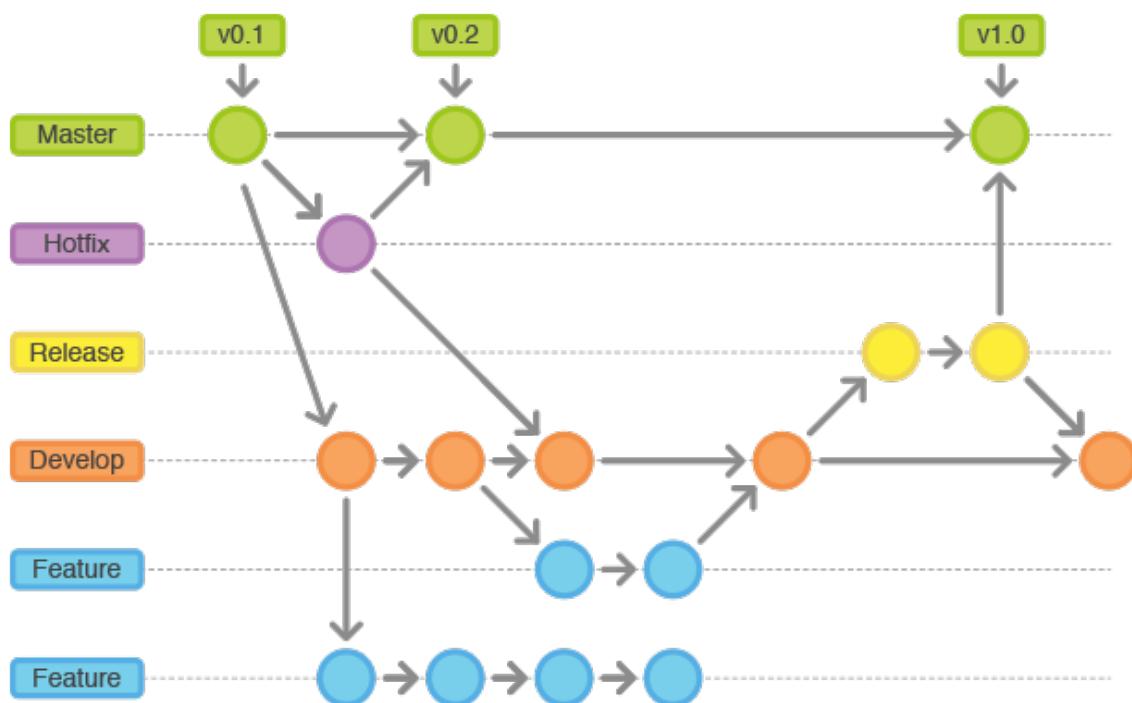


Figura 3.1: Esquema de git-flow

### Uso de comentarios en commits

Un commit es la acción de guardar o subir archivos a un repositorio. Los mensajes de commit son una herramienta muy útil en el desarrollo del software, ya que nos permiten encontrar cambios concretos en la historia de nuestro repositorio.

Cuando realizamos un commit, nos enfocamos en que el mensaje sea conciso y claro, de forma que brinde la suficiente información como para determinar qué cambios se realizaron en el commit.

## **Github board**

Para organizar el trabajo a realizar en cada uno de los *sprints*, hicimos uso de *Github board*. Con el uso de esta herramienta, todos los miembros del equipo tienen una noción del estado en que se encuentra el proyecto en todo momento, así como también, ayuda a lo que es la organización y priorización del trabajo a realizar.

### **Github board - Sprint 1**

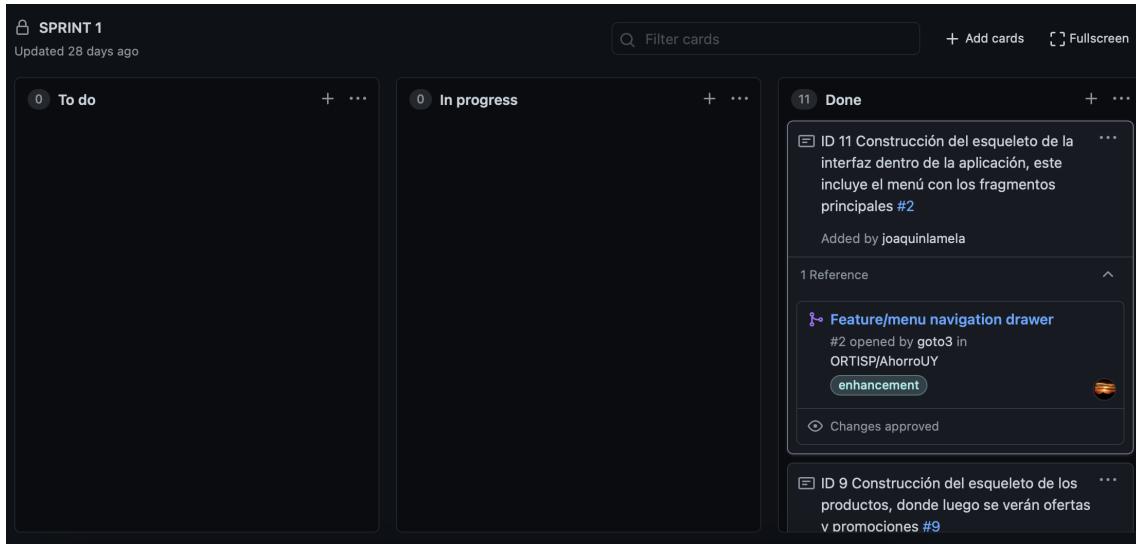


Figura 3.2: *Github board - Sprint 1*

### **Github board - Sprint 2**

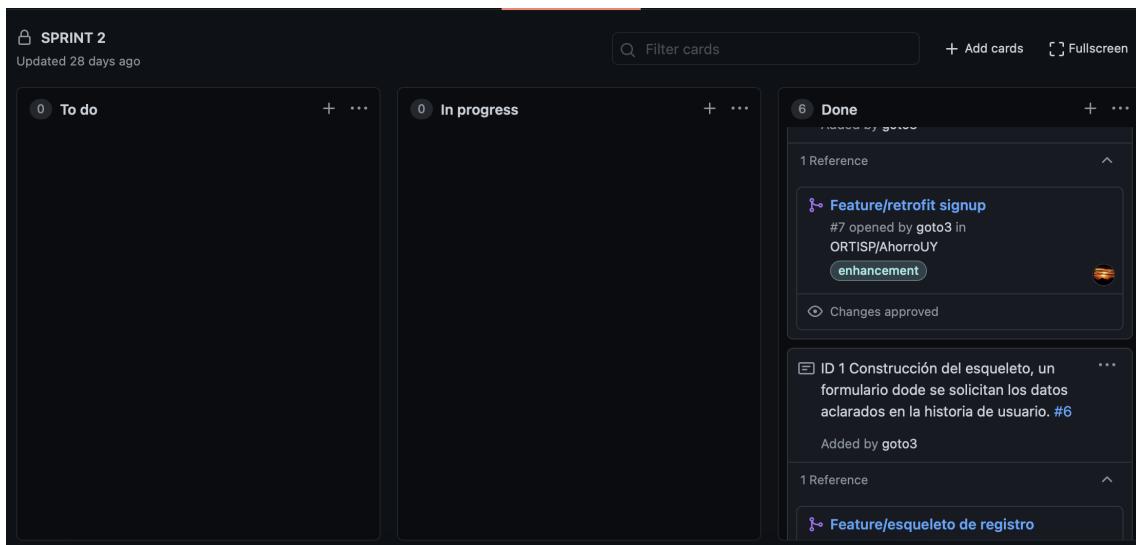


Figura 3.3: *Github board - Sprint 2*

## *Github board - Sprint 3*

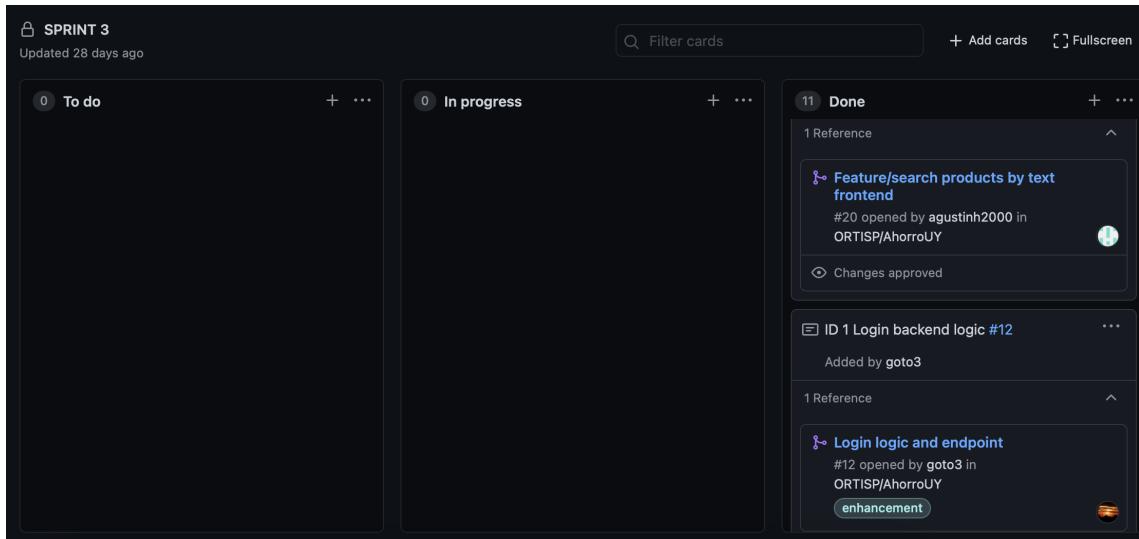


Figura 3.4: *Github board - Sprint 3*

## *Github board - Sprint 4*

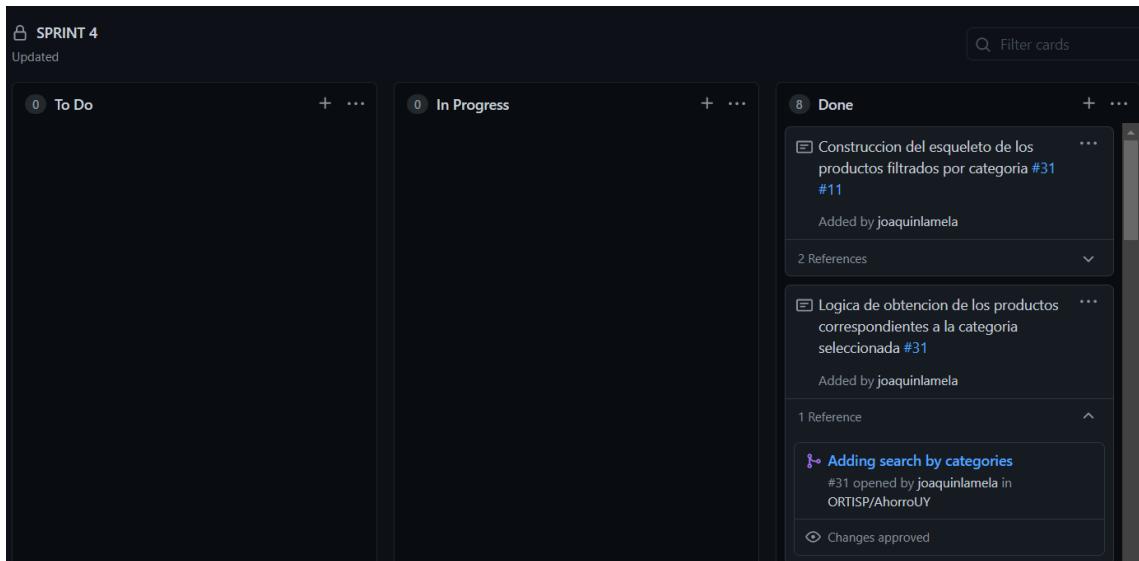


Figura 3.5: *Github board - Sprint 4*

## **Github board - Sprint 5**

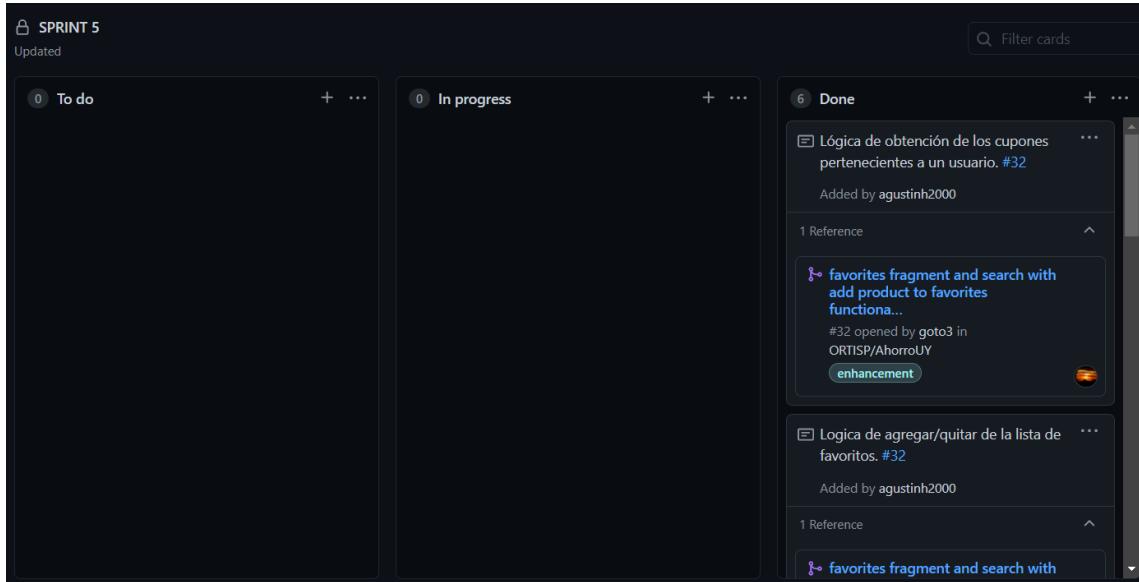


Figura 3.6: *Github board - Sprint 5*

## **Github board - Sprint 6**

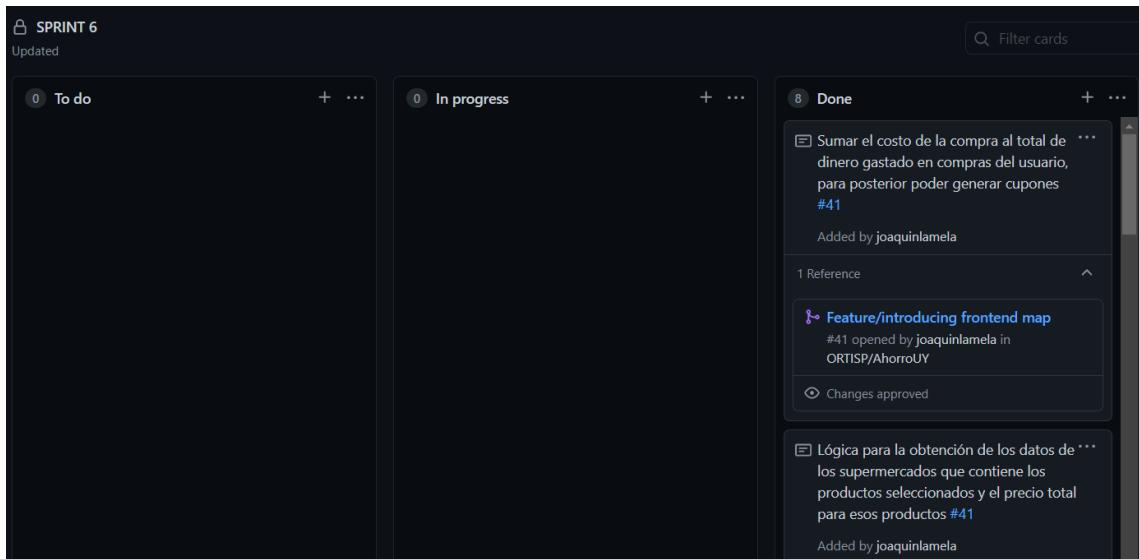


Figura 3.7: *Github board - Sprint 6*

## **Reporte de defectos en GitHub**

Los defectos encontrados a lo largo del proyecto, serán reportados mediante *issues* en *GitHub*. Un *issue* es la unidad de trabajo designada para realizar una mejora en un sistema informático.

En el título del reporte se indica el tipo de defecto: *compilation error*, *run time error* o *logic error*, junto al grado de severidad: *high*, *medium* o *low*, y una breve descripción del problema.

En el mensaje se indica que pasos se siguieron para llegar al error, para que el miembro del equipo que tenga que resolver el defecto, sepa en qué parte del sistema puede estar el error.

Luego cuando se soluciona el error el issue es cerrado y el que lo soluciona comenta que esto quedó solucionado, y comentar en que Pull Request fue solucionado.

En las siguientes figuras se pueden observar los *issues* reportados. Puede notarse como cada uno de ellos está identificado con una de las categorías mencionadas, así como también poseen un grado de severidad, para que de esa forma, los desarrolladores puedan saber cuales son los *issues* que hay que resolver con mayor urgencia. Además, debemos destacar que como se aprecia en la figura, hay un *issue* reportado que aún permanece abierto, porque dada su sencilla resolución y el bajo grado de severidad que posee, se decidió posergarlo con el fin de priorizar otros *issues* con mayor severidad.

The screenshot shows a list of 8 closed issues on a GitHub interface. The issues are listed vertically, each with a title, a severity label, and a small icon. The labels indicate the type of issue (enhancement or bug) and its logic level (LOW, MEDIUM). The severities range from LOW to HIGH. The issues are: 1. Additional cart feature [LOGIC - MEDIUM] enhancement (closed 5 days ago). 2. See the cart at all times in a reduced way [LOGIC - LOW] enhancement (closed 21 days ago). 3. Barcode scanning feature from menu[LOGIC - LOW] enhancement (closed 13 days ago). 4. Coupon functionality improvements - LOGIC [LOW] enhancement (closed 5 days ago). 5. It remains to add a criterion to the search, which is relevance. LOGIC [LOW] help wanted (closed 13 days ago). 6. Backend crash token [RUNTIME - HIGH] bug (closed on 13 May). 7. Behavior of the user session when re-opening the app LOGIC [MEDIUM] bug (closed 6 days ago). 8. Handling connection failures with API [RUNTIME - HIGH] bug (closed 4 days ago).

Figura 3.8: *Issues* cerrados.

The screenshot shows a list of issues on a GitHub interface. There is 1 open issue and 8 closed issues. The open issue is: Category icon behavior. LOGIC [LOW] enhancement, opened 23 days ago by goto3. The closed issues are identical to those in Figure 3.8.

Figura 3.9: *Issues* abiertos.

## Liberaciones

En cuanto a las liberaciones, en un principio el equipo había planificado realizar un total de 6 liberaciones a lo largo del proyecto, es decir, una liberación por *sprint*. Pero luego, sobre la marcha del proyecto, decidimos que esto no era muy factible debido a que estaríamos liberando versiones que serían muy poco utilizables, con funcionalidades muy reducidas y sin brindar una *UX* agradable.

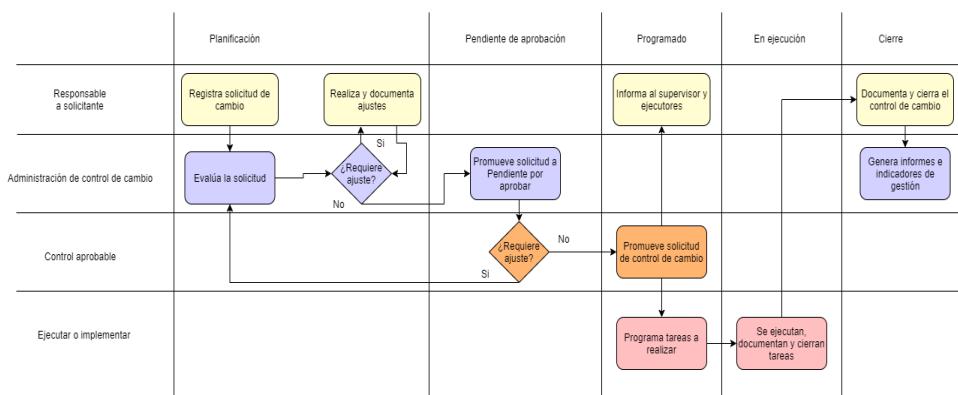
Entonces, decidimos como equipo, que la mejor decisión era realizar una liberación al final del proyecto, brindándole al usuario una versión estable, con una *UI* fluida que permita brindar una buena *UX*.

La liberación realizada en *GitHub*, se basa en la etiqueta *Git*, que permite marcar un punto específico en el historial del repositorio, y diferenciar esta versión de otras posteriores.

## 3.2. Proceso de gestión de solicitud de cambios

Se especificará en esta sección el proceso que se sigue y se seguirá para el trabajo con los cambios. El proceso es el siguiente.

El emisor solicita un cambio, o una actualización. Esta solicitud es revisada en una reunión de supervisión de cambios, en el cual se revisan los mismos, se planifican, se estiman los recursos, el riesgo, etc. En este momento, se tienen dos opciones para seguir, se puede confirmar, duplicar o rechazar el cambio. Esta información es enviada al cliente, quien reformula la solicitud y la reenvía. Una vez actualizada, nuevamente se presenta la reunión de supervisión del cambio, y en caso de querer continuar, se asigna y planifica el trabajo. Una vez hecho esto, se realiza el cambio, y el mismo se verifica. Se realizan todas las pruebas pertinentes, en caso necesario, se siguen realizando cambios, y una vez finalizado este proceso, se entrega el programa y se cierra la solicitud de cambio.



## 3.3. Herramientas de registro de defectos

Este análisis se realiza con una mirada en el código y no implica una ejecución del mismo. Se realizará este análisis, utilizando las *C# Coding Conventions*, que son los principales estándares para la correcta implementación de código en este lenguaje.

La misma permite también hacer un análisis del cumplimiento de las funcionalidades del programa, posibilidad de encontrar excepciones, fallas en la memoria, etc. Se utilizarán estos estándares debido a que se están continuamente actualizando hasta la fecha, y permiten un análisis completo y a fondo de las características mencionadas de un programa. Para poder detectar defectos, y posteriormente registrarlos utilizaremos las herramientas: *SonarQube* y *Designite*.

### 3.3.1. *SonarQube*

El programa nos muestra en resumen el resultado del análisis realizado, clasificando los defectos en distintas categorías:

- **Bugs:** Un error de codificación que romperá el código y debe corregirse de inmediato. (dominio de fiabilidad). El programa categoriza con una letra comprendida de la A a la E el nivel de fiabilidad que tiene nuestro sistema, con las siguientes reglas:
  - **A:** 0 errores.
  - **B:** Al menos 1 error menor. Estos son defectos de calidad que pueden afectar ligeramente la productividad del desarrollador: Por ej.: líneas de código demasiado largas.
  - **C:** Al menos 1 error mayor. Estos son defectos de calidad que pueden afectar la productividad del desarrollador: código descubierto, bloques duplicados, parámetros no utilizados.
  - **D:** Al menos 1 error crítico. Estos errores son aquellos que tienen una baja probabilidad de afectar el comportamiento de la aplicación en producción. Ej.: un bloque *catch* vacío, inyección de SQL.
  - **E:** Al menos 1 error bloqueador. Estos errores son aquellos que tienen una alta probabilidad de afectar el comportamiento de la aplicación en producción. En este caso el código debe corregirse de inmediato.
- **Vulnerabilities:** Una vulnerabilidad es un punto en el código que está abierto a ataques. El programa categoriza con una letra comprendida de la A a la E el nivel de seguridad que tiene nuestro sistema (de forma similar a como categoriza el nivel de fiabilidad).
- **Security Hotspots:** Resalta un fragmento de código sensible a la seguridad que el desarrollador debe revisar. Tras la revisión, descubrirá si no hay amenaza o que necesita aplicar una solución para proteger el código.
- **Maintainability:** El analizador divide esta sección en Code Smells y Technical Debt. El primero hace referencia a síntomas en el código fuente de un programa que posiblemente indican un problema más profundo, mientras que el segundo refiere al esfuerzo que debemos realizar para corregir todos los Code Smells en días, suponiendo un día de 8 horas.
- **Duplications:** El programa nos indica la cantidad de bloques de código duplicados, y el porcentaje de líneas duplicadas con respecto al total de líneas.

### 3.3.2. *Designite*

Designite es una herramienta de evaluación de la calidad del diseño de software. Analiza el código C# e identifica problemas de calidad del software. Específicamente, detecta un conjunto completo de *smells* en lo que refiere a arquitectura, diseño e implementación y proporciona mecanismos como análisis de métricas detalladas, matriz de estructura de dependencia, análisis de tendencias y mapas de distribución de *smells*. *Designite* ayuda a reducir la deuda técnica y mejorar la capacidad de mantenimiento del *software*.

#### Análisis con herramientas analizadoras de código

En primer lugar para cada uno de los paquetes de nuestro sistema realizamos un análisis de detección de *smells* en la arquitectura de la aplicación. Donde Designite detecta:

- Dependencia cíclica: este *smell* surge cuando dos o más componentes de la arquitectura dependen entre sí directa o indirectamente.
- Dependencia inestable: este *smell* surge cuando un componente depende de otros componentes que son menos estables que él.
- Interfaz ambigua: este *smell* surge cuando un componente ofrece solo un único punto de entrada general al componente.
- Componente 'de Dios': este *smell* ocurre cuando un componente es excesivamente grande en términos de LOC o número de clases.
- Concentración de características: este *smell* ocurre cuando un componente se da cuenta que realiza más de una funcionalidad arquitectónica.
- Funcionalidad dispersa: este *smell* surge cuando múltiples componentes son responsables de darse cuenta que realiza una misma responsabilidad.
- Estructura densa: este *smell* surge cuando los componentes tienen dependencias excesivas y densas sin ninguna estructura en particular.

Siendo entonces que pasaremos a realizar el análisis a los paquetes correspondientes.

#### WebApi

Podemos ver a partir del análisis realizado, que la arquitectura tiene un problema de *Dense structure* y otro de *Feature concentration*. Analizando en donde recaen los problemas marcados, podemos ver que los mismos son inevitables dada la complejidad de la solución en si, lo que genera entre otras cosas, que existan algunos paquetes que dependan de varios, aspecto que no es posible evitar. Lo que si realizamos, es que los paquetes dependan de otros más estables que él, así como también, que no dependan de implementaciones concretas y si de abstracciones bien definidas. Es por esto que no procederemos a realizar ninguna modificación sobre este paquete para

poder abstraer los problemas presentados ya que dada como se presenta la solución no es posible hacer cambios en ella.

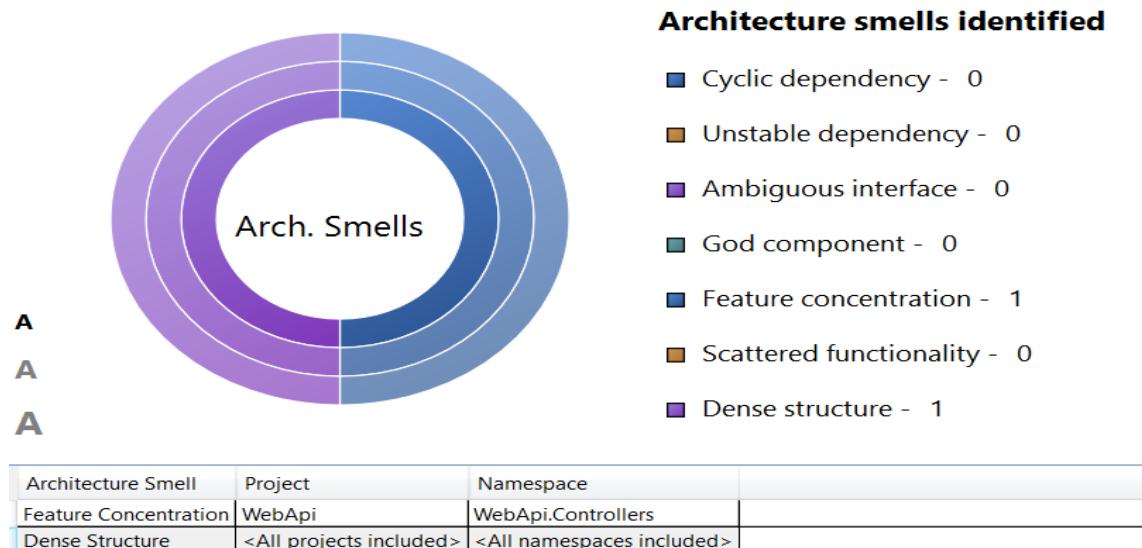


Figura 3.10: Análisis del paquete WebApi.

## BusinessLogic

Analizando este paquete podemos notar que el mismo presenta dos problemas, uno de *Feature concetration* y otro de *Scattered functionality*. Siendo que el primero de ellos, es correcto que se marque, dado que ninguno de los manejadores de las reglas de negocio deberían relacionarse entre ellos, ya que únicamente representan la operaciones relativas a las entidades que aparecen en sus nombres. Por otro lado, el segundo error, encontramos que también es correcto dado que nos indica que las clases que se encuentran muy relacionadas son los repositorios de usuario y la clase del dominio usuario, esto es 100 % correcto, ya que justamente representa el manejo de acceso a datos relativos a dicha clase del dominio. Es por esto que no procederemos a realizar ninguna modificación sobre este paquete para poder abstraer los problemas presentados ya que dada como se presenta la solución no es posible hacer cambios en ella.

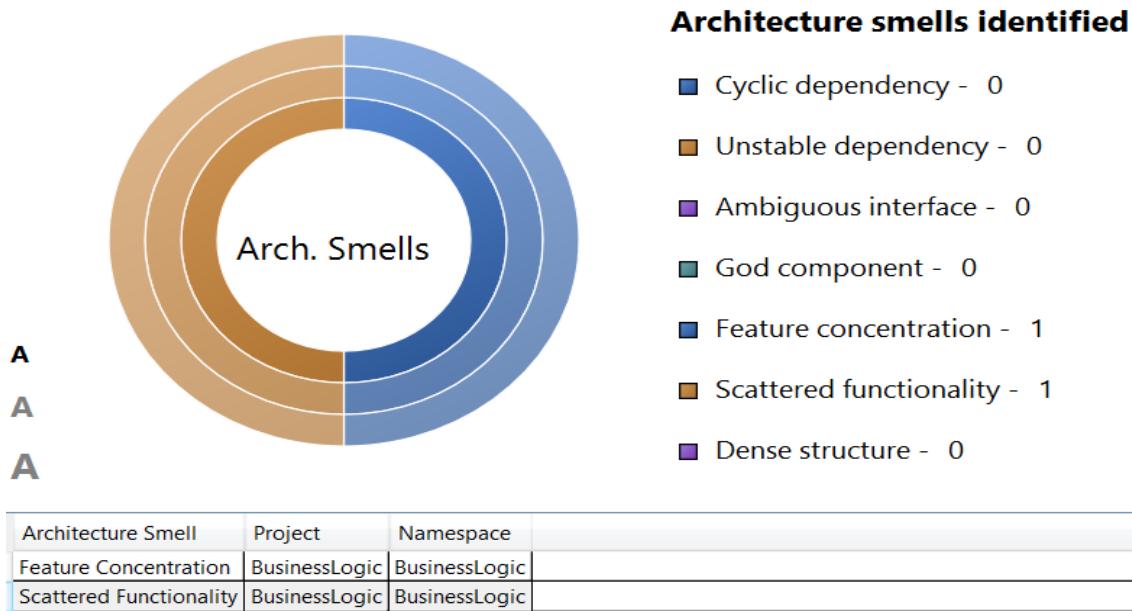


Figura 3.11: Análisis del paquete BusinessLogic.

## Domain

A partir del análisis realizado, podemos notar que la arquitectura del dominio tiene un problema de *Feature concetration*, esto ocurre debido a que la herramienta entiende que hay 3 tipos de conjuntos de clases dentro del dominio. Sucede que las clases que se encuentran en uno de los conjuntos son la mayor parte de las clases del dominio, esto dado que presentan relaciones entre ellas, y los restantes dos conjuntos son dos clases que no presentan relación (*Token.cs* y *UserSession.cs*). Analizando en donde recaen los problemas marcados, podemos ver que los mismos son inevitables dado a que el Token representa el token del teléfono móvil que llega desde la aplicación móvil para poder enviarle notificaciones, por lo cual es correcto que no tenga relación alguna con las clases del dominio. Y por otro lado, la clase UserSession también es correcta que se encuentre sin relacionarse con las otras clases del dominio ya que la misma posee el token de sesión relacionado con el identificador de un usuario, esto para que dicho token de sesión no tenga asociado credenciales de un usuario, únicamente su identificador. Es por esto que no procederemos a realizar ninguna modificación sobre este paquete para poder abstraer los problemas presentados ya que dada como se presenta la solución no es posible hacer cambios en ella.

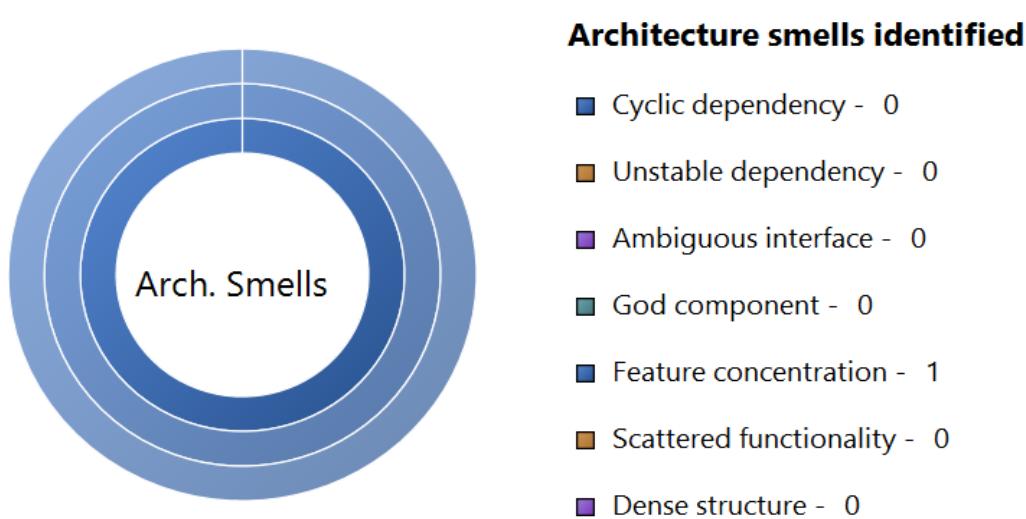


Figura 3.12: Análisis del paquete Domain.

## DataAccess

En el análisis de este paquete, la herramienta detecta dos *Feature concentration*. Viendo en detalle, uno de ellos esta relacionado con lo que son las migraciones de base de datos, por lo cual no sería un problema de la solución. El restante problema encontrado, ocurre por un aspecto similar al que se nombró en el paquete anterior, es decir, tenemos clases en este paquete que no se relacionan con las restantes clases del paquete como es el caso de: ContextFactory y ContextObl. Es correcto que no se relacionen con las demás debido a que se encargan de manejar el contexto de la base de datos, y es razonable que no se relacionen con las demás entidades. Por lo mencionado, no procederemos a realizar ninguna modificación sobre este paquete para poder abstraer los problemas presentados ya que dada como se presenta la solución no es posible hacer cambios en ella.

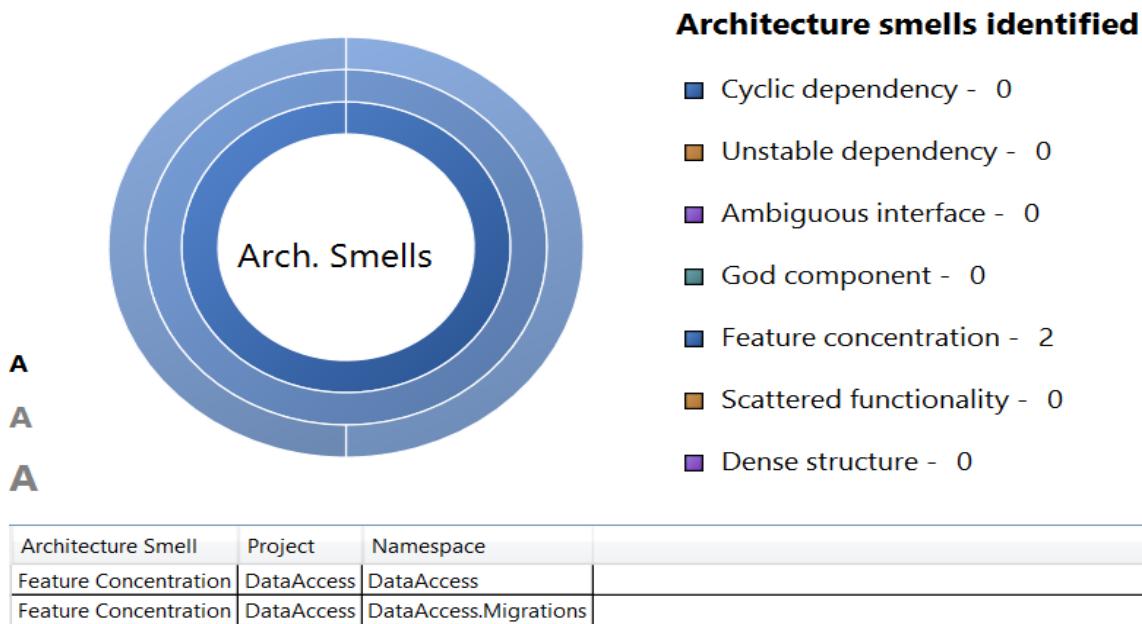


Figura 3.13: Análisis del paquete DataAccess.

Continuando con el análisis pasaremos a realizarlo sobre las métricas de la estructura (*Compute metrics*). Esto dado que Designite calcula métricas de diseño orientadas a objetos que son útiles para medir la salud estructural del proyecto de software. Designite clasifica estas métricas en cuatro categorías: métricas a nivel de solución, métricas a nivel de proyecto, métricas a nivel de método y métricas a nivel de clase. Designite resalta valores de métricas específicas que violan el umbral de métrica como una violación de métrica en un color diferente para ayudarlo a detectar las violaciones. Entonces nuevamente realizaremos el análisis para cada uno de los paquetes ya presentados.

En particular nos centraremos en la métrica conocida como WMC, la cual significa *Weighted Method per Class*, cuyo sentido creemos que es la más importante a la hora de analizar además de la complejidad ciclomática. Esta métrica es la suma de las complejidades de los métodos definidos en una clase. Por lo tanto, representa la complejidad de una clase en su conjunto y esta medida se puede utilizar para indicar el esfuerzo de desarrollo y mantenimiento de la clase.

En todos los casos, se puede observar que todos los paquetes tienen una muy buena medida de esta métrica, observando que se obtienen los resultados en color verde, clasificando en todos los casos que son: 'clases que respetan el umbral métrico'.

## WebApi

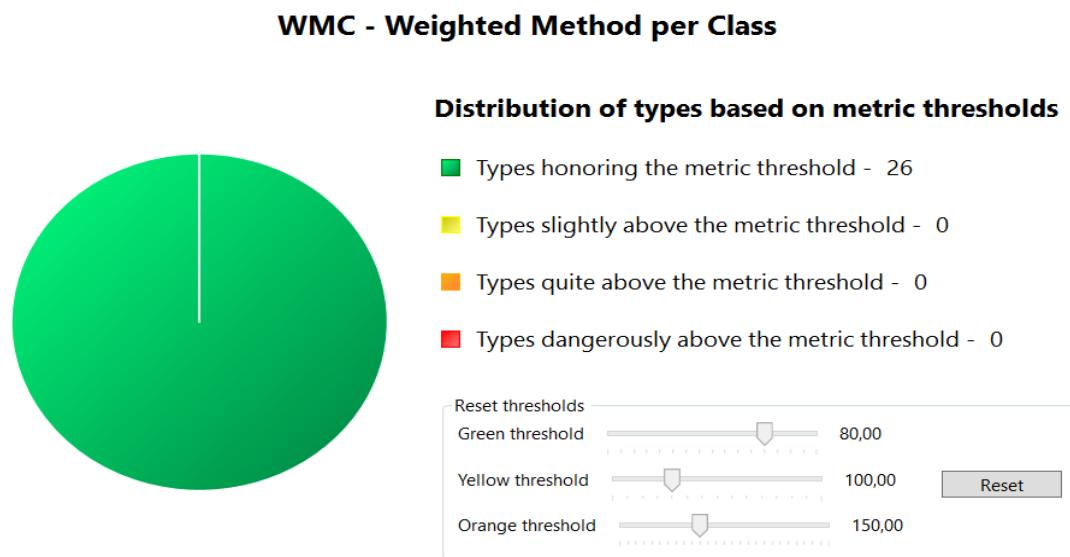


Figura 3.14: Análisis del paquete WebApi.

## BusinessLogic

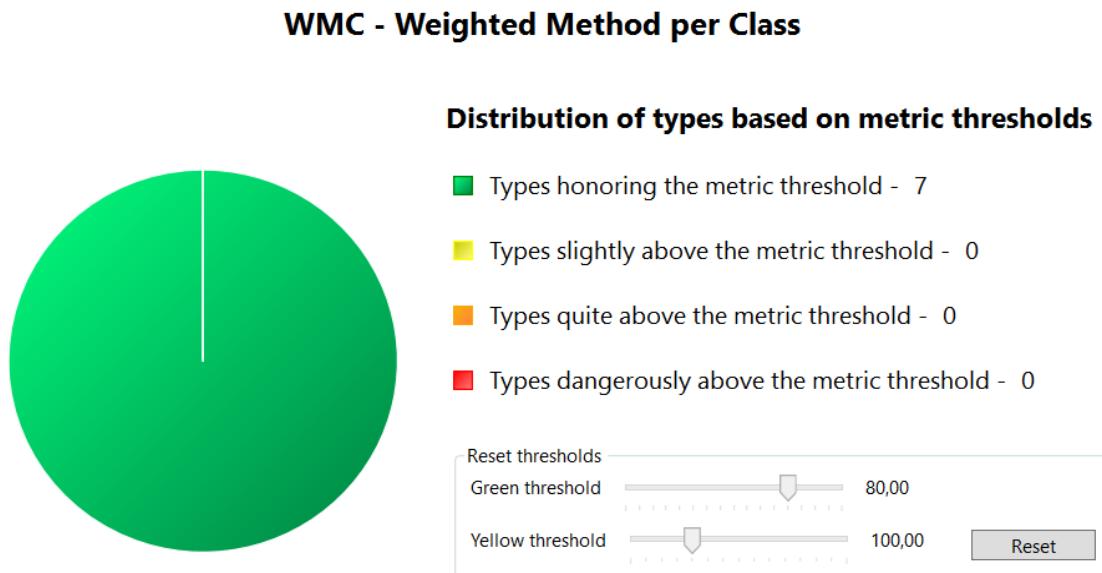


Figura 3.15: Análisis del paquete BusinessLogic.

## Domain

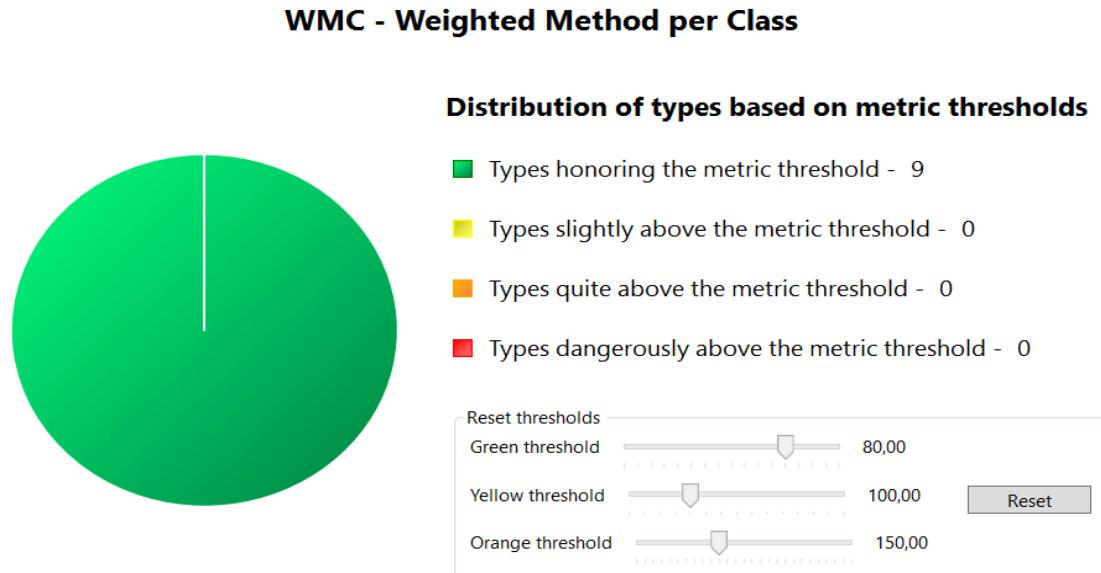


Figura 3.16: Análisis del paquete Domain.

## DataAccess

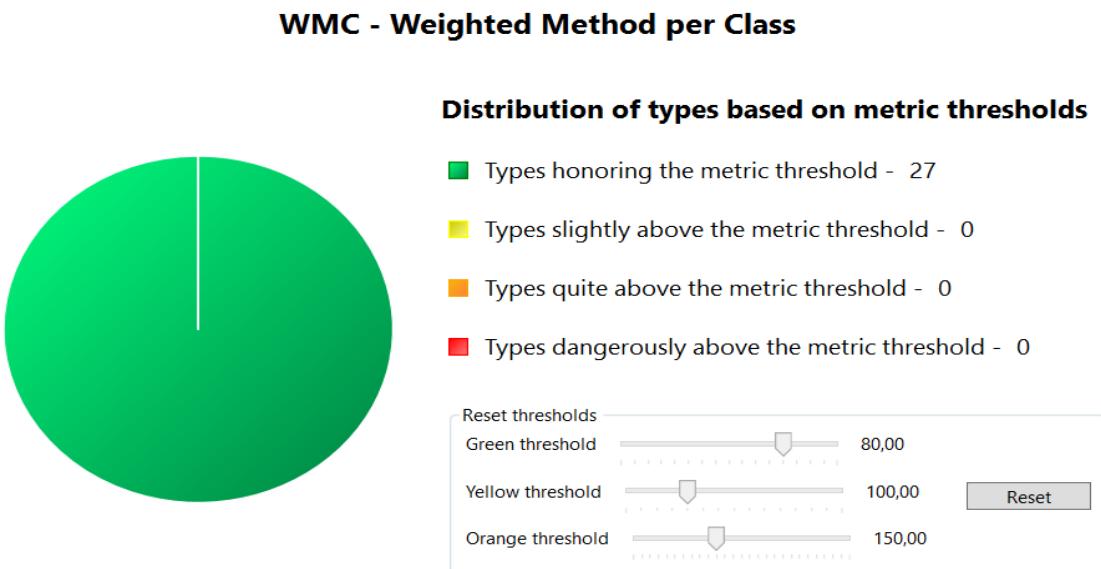


Figura 3.17: Análisis del paquete DataAccess.

Por otro lado encontramos también el análisis respectivo a la complejidad cíclomática, donde encontramos que en todos los casos, los paquetes tienen una muy buena medida de esta métrica, observando que se obtienen los resultados en color verde, clasificando en todos los casos que: 'las clases respetan el umbral métrico'. Salvo en el caso del paquete de la WebApi donde encontramos que uno de ellos está en color rojo dada su complejidad. Esto ocurre dado a que es la clase que se encarga de especificar los errores para poner los mensajes y código de estado, esto no es posible cambiarlo dado a que toda la aplicación se maneja con esto.

## WebApi

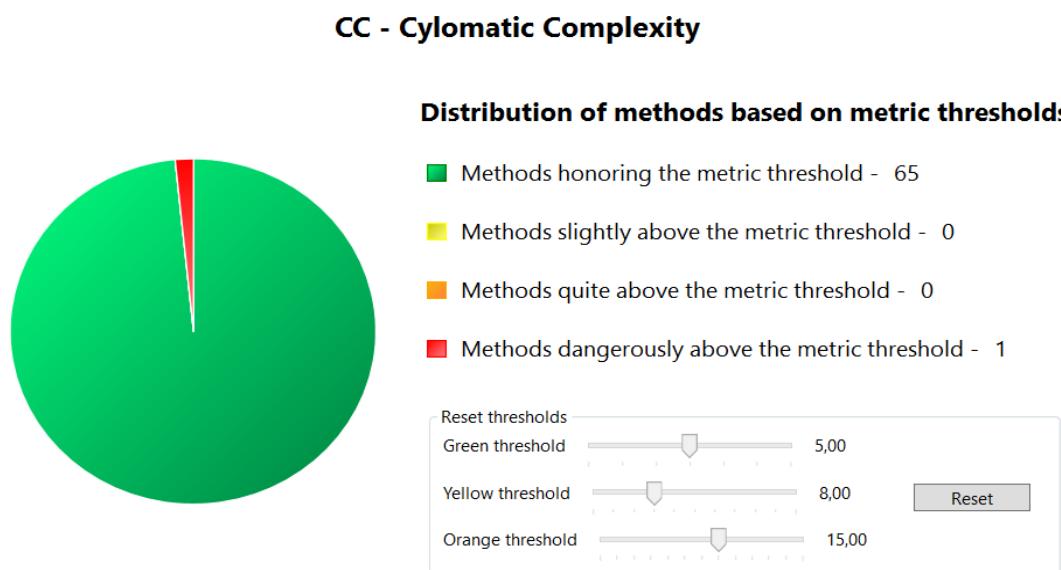


Figura 3.18: Análisis del paquete WebApi.

## BusinessLogic

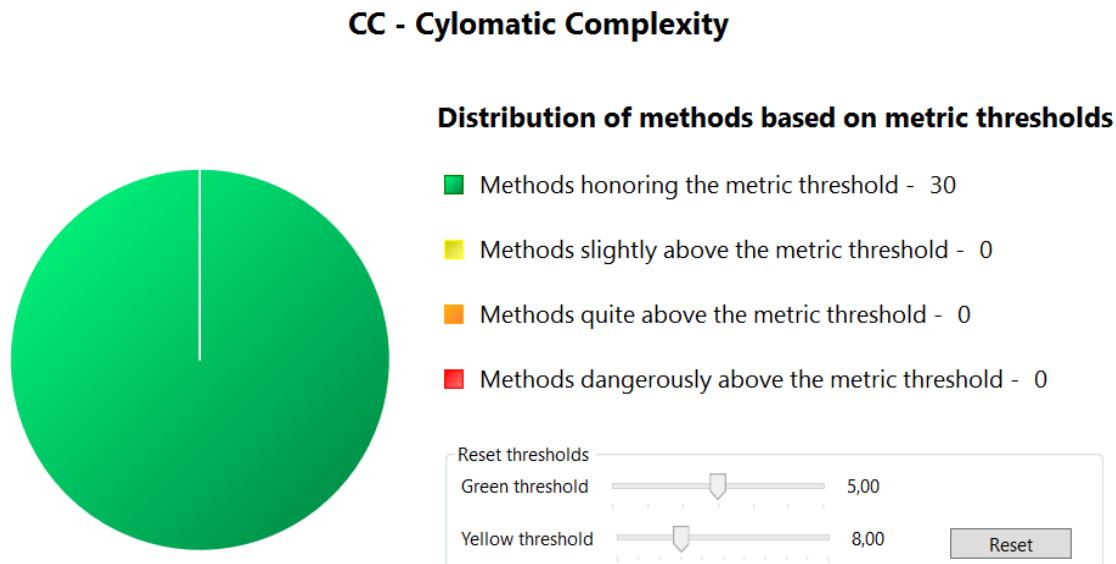


Figura 3.19: Análisis del paquete BusinessLogic.

## Domain

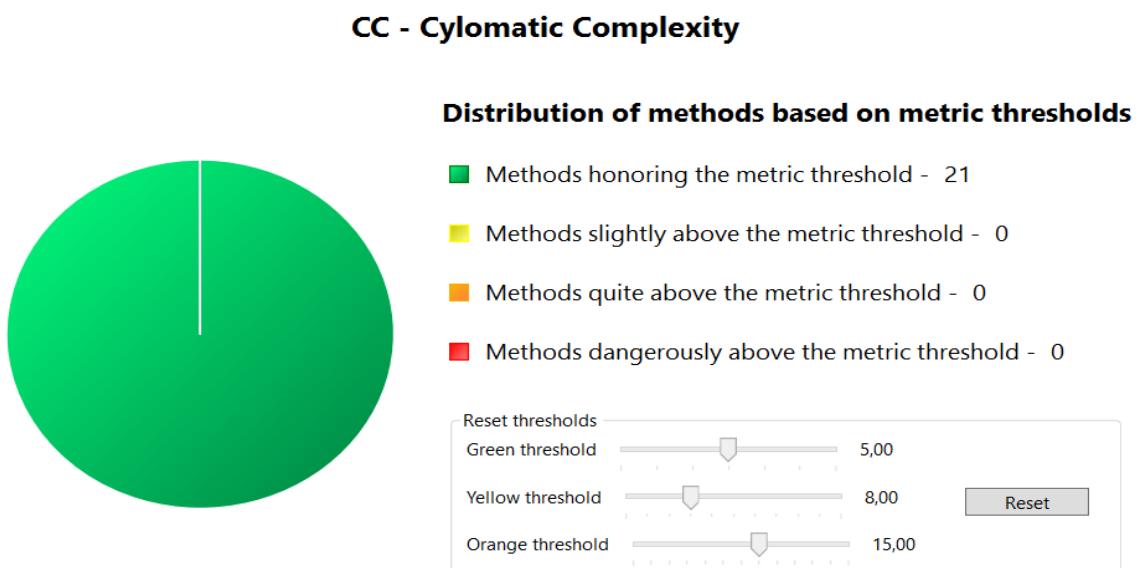


Figura 3.20: Análisis del paquete Domain.

## DataAccess

### CC - Cylomatic Complexity

#### Distribution of methods based on metric thresholds

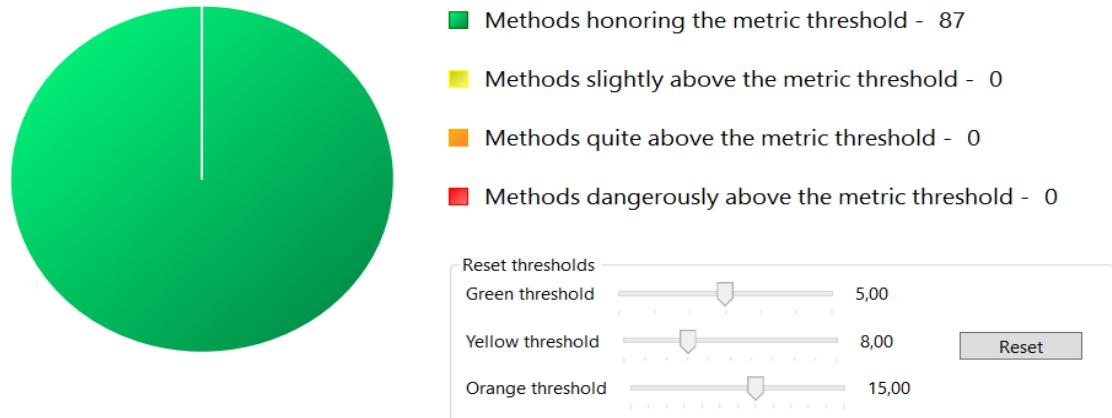


Figura 3.21: Análisis del paquete DataAccess.

## 3.4. Pruebas de *software*

Realizamos *tests* para cada una de las capas de nuestra aplicación, es decir, tenemos un paquete de *tests* para la *WebApi*, otro para la *BusinessLogic* y otro para el paquete *DataAccess*.

A su vez, dentro de cada uno de estos paquetes de *tests* tenemos diferentes clases, que se utilizan para probar cada una de las entidades. En el paquete *WebApiTest*, tenemos una clase de prueba por cada *controller* de la *WebApi*, en el de *BusinessLogicTest* consideramos apropiado realizar una clase de *test* para cada clase manejadora del paquete *BusinessLogic*, y de la misma forma, en el paquete *DataAccessTest* tenemos una clase de *test* por cada clase del paquete *DataAccess*. Esto lo hicimos con el objetivo de que los *tests* que prueban operaciones de una determinada clase estén todos juntos, y que sean independientes de otros que prueban otros tipos de objetos.

En particular se tomó esta decisión para seguir las características que deben poseer las pruebas unitarias, conocido como el principio FIRST, definido por el autor *Robert Cecil Martín*.

El principio FIRST, es el acrónimo de las cinco características que deben tener nuestros *tests* unitarios para ser considerados *tests* con calidad. Siendo las características:

- **F (fast-rápido):** posibilidad de ejecutar un gran número de *tests* en cuestión de segundos.
- **I (independent-independiente):** todas las pruebas unitarias deben de ser independientes de las otras. En el momento que un *test* falla por el orden en el que se ha ejecutado, este *test* está mal desarrollado. El resultado no debe verse alterado ejecutando los *tests* en un orden y otro o incluso de forma independiente.
- **R (repeatable-repetible):** El resultado de las pruebas debe ser el mismo independientemente del pc/servidor/terminal en el que se ejecute.
- **S (self-validating- auto evaluable):** La ventaja de las pruebas automatizadas es que podemos ejecutarlas simplemente al pulsar un botón o incluso hacer que se ejecuten de forma automática tras otro proceso.
- **T (timely-oportuno):** Las pruebas unitarias deben escribirse justo antes del código de producción que las hace pasar. Si se escriben pruebas después del código de producción, se puede encontrar que el código de producción es difícil de probar.

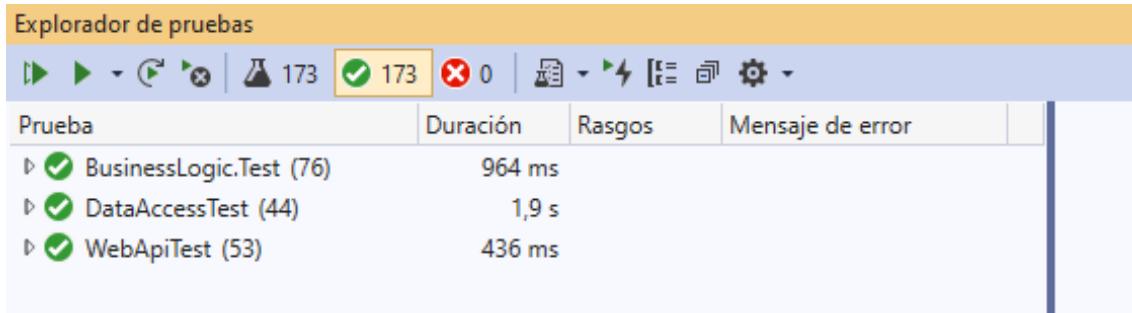
### 3.4.1. Cobertura de pruebas

Utilizando el explorador de pruebas, podemos apreciar que se realizaron un total de 173 *tests*, obteniendo un porcentaje de pruebas exitosas igual al 100

Esos 173 *tests* se dividen de la siguiente forma:

- 76 pertenecen al paquete *BusinessLogicTest*.

- 44 pertenecen al paquete *DataAccessTest*.
- 53 pertenecen al paquete *WebApiTest*.



Si pasamos a ejecutar el analizador de cobertura de código, obtenemos finalmente que el porcentaje total de cobertura para toda la solución es igual al 97,24 %. Esta cobertura es gracias a que una vez que se había desarrollado la funcionalidad se realizaban los test para verificar su correcto funcionamiento.

Joaquin Lamela_DESKTOP-LA054R4 2021-01				
Jerarquía	No cubiertos (bloques)	No cubiertos (% de bloques)	Cubiertos (bloques)	Cubiertos (% de bloques)
▶ Joaquin Lamela_DESKTOP-LA... 180	2,76 %	6342	97,24 %	
▷ businesslogic.dll 0	0,00 %	208	100,00 %	
▷ businesslogic.test.dll 56	2,20 %	2490	97,80 %	
▷ businesslogicexception.dll 0	0,00 %	9	100,00 %	
▷ businesslogicforpushnotif... 1	3,13 %	31	96,88 %	
▷ dataaccess.dll 40	5,81 %	649	94,19 %	
▷ dataaccesstest.dll 17	2,63 %	629	97,37 %	
▷ domain.dll 14	5,74 %	230	94,26 %	
▷ domainexception.dll 0	0,00 %	3	100,00 %	
▷ entitiesforpushnotificatio... 0	0,00 %	10	100,00 %	
▷ repositoryexception.dll 0	0,00 %	9	100,00 %	
▷ webapi.dll 12	2,06 %	571	97,94 %	
▷ webapimodels.dll 4	7,55 %	49	92,45 %	
▷ webapitest.dll 36	2,42 %	1454	97,58 %	

### 3.4.2. *BusinessLogic*

Pasando a analizar el paquete *BusinessLogic*, podemos notar que el porcentaje de cobertura de líneas de código del mismo es igual al 100 %, esto quiere decir, que cada línea de las clases pertenecientes a *BusinessLogic* fueron testeadas.

▶ businesslogic.dll	0	0,00 %	208	100,00 %
◀ BusinessLogic	0	0,00 %	208	100,00 %
▷ CategoryManage... 0	0,00 %	9	100,00 %	
▷ CouponManagem... 0	0,00 %	32	100,00 %	
▷ FavoritesManagem... 0	0,00 %	31	100,00 %	
▷ FavoritesManagem... 0	0,00 %	3	100,00 %	
▷ ProductMarketMan... 0	0,00 %	43	100,00 %	
▷ PurchaseManagem... 0	0,00 %	27	100,00 %	
▷ UserManagement 0	0,00 %	34	100,00 %	
▷ UserSessionManag... 0	0,00 %	29	100,00 %	

### 3.4.3. BusinessLogicException

En este paquete, podemos notar que el porcentaje de cobertura es igual al 100 %, es decir, todas las excepciones de las reglas de negocio fueron testeadas por completo y de forma exitosa.

businesslogicexception.dll	0	0,00 %	9	100,00 %
BusinessLogicException	0	0,00 %	9	100,00 %
ClientBusinessLogi...	0	0,00 %	6	100,00 %
DomainBusinessLo...	0	0,00 %	2	100,00 %
MessageException...	0	0,00 %	1	100,00 %

### 3.4.4. BusinessLogicTest

Podemos apreciar que el porcentaje de cobertura de *BusinessLogicTest* es igual a 97,80 %. Este análisis lo hacemos básicamente para saber si tenemos ciertas líneas en los tests que no son necesarias dejarlas porque no se están ejecutando, en este caso, al ingresar a revisar qué es lo que está sucediendo, notamos que no es que existan líneas que no se están ejecutando, sino que considera a las llaves “{ “ y “ }” como parte del código y no lo está contabilizando cuando analiza la cobertura.

businesslogic.test.dll	56	2,20 %	2490	97,80 %
BusinessLogic.Test	56	2,20 %	2490	97,80 %
CategoryTest	2	2,60 %	75	97,40 %
CouponTest	11	1,83 %	589	98,17 %
FavoritesTest	12	1,84 %	641	98,16 %
ProductMarketTest	12	2,94 %	396	97,06 %
PurchaseTest	1	0,62 %	161	99,38 %
PushNotificationTest	1	2,08 %	47	97,92 %
UserSessionTest	5	2,21 %	221	97,79 %
UserTest	12	3,23 %	360	96,77 %

Esto se puede apreciar en la siguiente imagen, donde claramente se puede observar que la llave que cierra al método de testeо, no está siendo contabilizada por el analizador de código.

```
[TestMethod]
[ExpectedException(typeof(ServerException))]

public void GetAllCouponsTestInternalServerErrorToken()
{
    Guid token = new Guid();

    var userRepositoryMock = new Mock<IUserRepository>(MockBehavior.Strict);
    userRepositoryMock.Setup(m => m.GetUserByIdWithCoupons(It.IsAny<Guid>())).Returns(userWithCoupons);
    var userSessionRepositoryMock = new Mock<IUserSessionRepository>(MockBehavior.Strict);
    userSessionRepositoryMock.Setup(m => m.GetTokenUserId(It.IsAny<Guid>())).Throws(new ServerException());
    var marketRepositoryMock = new Mock< IRepository<Market>>(MockBehavior.Strict);

    ICouponManagement couponManagement = new CouponManagement(userRepositoryMock.Object
        , marketRepositoryMock.Object, userSessionRepositoryMock.Object);

    couponManagement.GetAllFromUser(token);
}
```

Cómo se puede observar, lo que se muestra en rojo, son líneas las cuales no son alcanzadas por el analizador de cobertura, de tal manera que nos está queriendo

decir que esa llave no llega a ser probada, lo cual no es cierto debido a que el test es ejecutado con normalidad. Por lo cual, ese resultado de cobertura es un tanto engañoso debido a que todos los test se ejecutan correctamente y cubren todas las líneas de los mismos.

### 3.4.5. *DataAccess*

Analizando el porcentaje de cobertura del paquete *DataAccess*, se obtiene que el mismo es igual a 94,19 %. Analizando los métodos particulares en que la cobertura no llega al 100 %, notamos que esto se debe a que dentro de estos métodos se está capturando una *Exception* con el fin de que si ocurre un error interno en la base de datos, como puede ser una desconexión, el sistema pueda capturar esa excepción e informar al usuario que es lo que ha sucedido. Lo que sucede es que este tipo de excepciones, al surgir por un error inesperado de la base de datos, no es posible probarlas desde los métodos de tests (a no ser que se pruebe con mocking, pero no es adecuado para las pruebas de acceso a datos), y eso implica que el porcentaje de cobertura descienda. Cabe destacar que aquí, se utilizó el tag [ExcludeFromCodeCoverage] con el fin de que no se tomen en cuenta las migraciones a la hora de realizar el análisis de cobertura de código.

▲	dataaccess.dll	40	5,81 %	649	94,19 %
◀	{ DataAccess	40	5,81 %	649	94,19 %
▷	BaseRepository<T>	10	13,70 %	63	86,30 %
▷	ContextFactory	0	0,00 %	9	100,00 %
▷	ContextObl	9	9,18 %	89	90,82 %
▷	ProductMarketRep...	10	3,31 %	292	96,69 %
▷	ProductMarketRep...	0	0,00 %	35	100,00 %
▷	ProductMarketRep...	0	0,00 %	3	100,00 %
▷	ProductMarketRep...	1	3,57 %	27	96,43 %
▷	UserRepository	8	7,02 %	106	92,98 %
▷	UserSessionReposit...	2	7,41 %	25	92,59 %

En esta imagen, se puede apreciar como el catch de la *Exception*, no se encuentra testeado por lo descrito previamente.

```
8 referencias | 6/6 pasando | Joaquin, Hace 23 días | 2 autores, 2 cambios
public List<Product> GetProductsAvailablesInMarkets()
{
    if (Context.Database.CanConnect())
    {
        List<ProductMarket> productsAvailableInMarkets = context.Set<ProductMarket>().ToList();
        List<Product> distinctProducts = productsAvailableInMarkets
            .GroupBy(p => p.ProductId)
            .Select(p => p.First())
            .ToList().ConvertAll(p => p.Product);
        if (distinctProducts.IsNullOrEmpty())
        {
            throw new ClientException(RepositoryMessagesException.ErrorNotFoundProducts);
        }
        return distinctProducts;
    }
    else
    {
        throw new ServerException(RepositoryMessagesException.ErrorConnectingIntoDatabase);
    }
}
```

### 3.4.6. *DataAccessTest*

Analizando la cobertura del paquete *DataAccessTest* notamos que la misma es igual al 97.37 %. La razón por la cual no se cubre un 100 % se debe a la misma razón que la del paquete *BusinessLogicTest* (está contabilizando como que las “{“ y “}” no se están ejecutando).

▲	dataaccesstest.dll	17	2,63 %	629	97,37 %
▲	DataAccessTest	17	2,63 %	629	97,37 %
▷	CategoryTest	1	4,35 %	22	95,65 %
▷	MarketTest	1	3,13 %	31	96,88 %
▷	ProductMarketTest	9	2,33 %	378	97,67 %
▷	ProductTest	3	4,48 %	64	95,52 %
▷	UserSessionTest	0	0,00 %	27	100,00 %
▷	UserTest	3	2,73 %	107	97,27 %

Esto se puede apreciar en la siguiente imagen, donde claramente se puede observar que la llave que cierra al método de testeo, no está siendo contabilizada por el analizador de código.

```
[TestMethod]
[ExpectedException(typeof(ClientException))]
| 0 referencias | Joaquin, Hace 31 días | 1 autor, 1 cambio
public void TestGetProductNotFoundByBarcode()
{
    ContextObl context = ContextFactory.GetMemoryContext(Guid.NewGuid().ToString());
    IProductMarketRepository productMarketRepository = new ProductMarketRepository(context);
    productMarketRepository.Add(aProductMarket);
    productMarketRepository.GetProductByBarcode("123");
}
```

De tal forma como se puede observar, lo que se muestra en rojo, son líneas las cuales no son alcanzadas por el analizador de cobertura, de tal manera que nos está queriendo decir que esa llave no llega a ser probada, lo cual no es cierto debido a que el test es ejecutado con normalidad. Por lo cual, ese resultado de cobertura es un tanto engañoso debido a que todos los test se ejecutan correctamente y cubren todas las líneas de los mismos.

### 3.4.7. *Domain*

Analizando la cobertura del paquete *Domain* podemos observar que la misma es igual al 94,26 %. La razón por la cual no se llega al 100 % de cobertura, se basa en que no fueron probados, algunas secciones de métodos equals, específicamente en aquellos en que se recibe un objeto null por parámetro, luego restan probar algunos métodos get y set principalmente referidos a las clases que representan las relaciones entre Productos, Usuarios y Markets respectivamente, entonces los atributos de id, no son muy utilizados en las pruebas a nivel de lógica de negocios o dominio, debido a que el propósito que tienen dichos atributos es referido al nivel de base de datos.

domain.dll	14	5,74 %	230	94,26 %
Domain	14	5,74 %	230	94,26 %
Category	1	6,25 %	15	93,75 %
Coupon	0	0,00 %	10	100,00 %
Market	2	4,08 %	47	95,92 %
Product	4	14,29 %	24	85,71 %
ProductMarket	1	3,85 %	25	96,15 %
Purchase	3	10,34 %	26	89,66 %
Token	1	8,33 %	11	91,67 %
User	1	1,56 %	63	98,44 %
UserSession	1	10,00 %	9	90,00 %

### 3.4.8. *DomainException*

Analizando la cobertura de *DomainException*, se observa que la misma es igual a 100 %, es decir, que todas las excepciones relativas al dominio fueron testeadas.

domainexception.dll	0	0,00 %	3	100,00 %
DomainException	0	0,00 %	3	100,00 %
MessageException...	0	0,00 %	1	100,00 %
UserException	0	0,00 %	2	100,00 %

### 3.4.9. *BusinessLogicForPushNotification*

Analizando la cobertura del paquete *BusinessLogicForPushNotification* podemos observar que la misma es igual al 96,68 %. La razón por la cual no se llega al 100 % de cobertura, se basa en que la operación de enviar notificaciones tiene un operador binario O, el cual indica si se envió o no. El caso de no enviarlo no se puede emanar.

businesslogicforpushnotif...	1	3,13 %	31	96,88 %
BusinessLogicForPush...	1	3,13 %	31	96,88 %
PushNotificationM...	1	3,13 %	31	96,88 %

### 3.4.10. *EntitiesForPushNotification*

Pasando a analizar el paquete *EntitiesForPushNotification*, podemos notar que el porcentaje de cobertura de líneas de código del mismo es igual al 100 %, esto quiere decir, que cada línea de las clases pertenecientes a *EntitiesForPushNotification* fueron testeadas.

entitiesforpushnotificatio...	0	0,00 %	10	100,00 %
EntitiesForPushNotific...	0	0,00 %	10	100,00 %
Message	0	0,00 %	6	100,00 %
Notification	0	0,00 %	4	100,00 %

### 3.4.11. *RepositoryException*

Se puede apreciar que el porcentaje de cobertura del paquete *RepositoryException* es igual a 100 %. A partir de esto, podemos indicar que todas las excepciones que fueron creadas para ser lanzadas cuando ocurre un error en la base de datos, están probadas por completo.

repositoryexception.dll	0	0,00 %	9	100,00 %
RepositoryException	0	0,00 %	9	100,00 %
ClientException	0	0,00 %	4	100,00 %
RepositoryMessage	0	0,00 %	1	100,00 %
ServerException	0	0,00 %	4	100,00 %

### 3.4.12. WebApi

Analizando la cobertura de *WebApi* podemos observar que la misma asciende al 97,24 %. La razón por la cual no se llega a un 100 % de cobertura es debido a que dentro de este *namespace* también se encuentran algunos DTOs que contienen algunos gets y sets, como también algunos métodos *equals*, los cuales no fueron testeados.

Cabe aclarar que en este paquete se utilizó el tag [ExcludeFromCodeCoverage] con el fin de que no se tomen en cuenta las clases Program y Startup a la hora de realizar el análisis de cobertura de código.

webapi.dll	12	2,06 %	571	97,94 %
WebApi.Controllers	0	0,00 %	224	100,00 %
WebApi.DataTypes.Fo...	0	0,00 %	10	100,00 %
WebApi.DataTypes.Fo...	0	0,00 %	14	100,00 %
WebApi.DataTypes.Fo...	0	0,00 %	20	100,00 %
WebApi.DataTypes.Fo...	3	12,50 %	21	87,50 %
WebApi.DataTypes.Fo...	2	5,26 %	36	94,74 %
WebApi.DataTypes.Fo...	4	4,04 %	95	95,96 %
WebApi.DataTypes.Fo...	2	1,68 %	117	98,32 %
WebApi.DataTypes.Fo...	0	0,00 %	2	100,00 %
WebApi.DataTypes.Fo...	1	3,03 %	32	96,97 %

### 3.4.13. WebApiTest

Analizando la cobertura del paquete *WebApiTest* podemos ver que la misma es igual al 97,58 %. La razón por la cual no se cubre un 100 % se debe a la misma razón que la del paquete *BusinessLogicTest* (está contabilizando como que las “{” y “}” no se están ejecutando).

webapitest.dll	36	2,42 %	1454	97,58 %
WebApiTest	36	2,42 %	1454	97,58 %
BestOptionControl...	1	0,96 %	103	99,04 %
CategoryController...	2	2,47 %	79	97,53 %
CouponsController...	2	2,15 %	91	97,85 %
FavoritesController...	9	2,13 %	413	97,87 %
ProductMarketCon...	8	2,01 %	391	97,99 %
PurchaseController...	3	5,77 %	49	94,23 %
SessionControllerT...	4	3,67 %	105	96,33 %
TokenControllerTest	1	2,38 %	41	97,62 %
UserControllerTest	6	3,19 %	182	96,81 %

En el siguiente código el cual es un test que se encuentra dentro del paquete, podemos observar lo sucedido con las llaves.

```
[TestMethod]
[ExpectedException(typeof(ServerException))]
➊ | 0 referencias | Joaquin, Hace 21 días | 1 autor, 1 cambio
public void GetAllCouponsFromUserInternalServerErrorTest()
{
    List<Coupon> listOfCoupon = new List<Coupon>() { coupon };

    var mock = new Mock<ICouponManagement>(MockBehavior.Strict);
    mock.Setup(m => m.GetAllFromUser(It.IsAny<Guid>())).Throws(new ServerException());
    CouponsController couponController = new CouponsController(mock.Object);
    var result = couponController.GetAllFromUser(Guid.NewGuid().ToString());
}
```

# 4. Decisiones de Diseño

## 4.1. Diseño de la arquitectura de la aplicación móvil

Para diseñar la arquitectura de la aplicación móvil, se siguieron las prácticas y arquitecturas recomendadas que permiten el desarrollo de aplicaciones sólidas y de calidad.

### 4.1.1. Descripción General - MVVM Architecture

En la figura que se muestra a continuación, se observa una arquitectura general de como se ha desarrollado la aplicación móvil, la cual se conoce como *MVVM* (*Model-View-ViewModel*). Se puede observar como cada componente solo depende del componente que está un nivel más abajo (solo tienen referencia por observables). Por ejemplo, las actividades y los fragmentos solo dependen de un modelo de vista. El repositorio es la única clase que depende de otras clases. En nuestra aplicación, el repositorio depende una fuente de datos proveniente desde el *backend*.

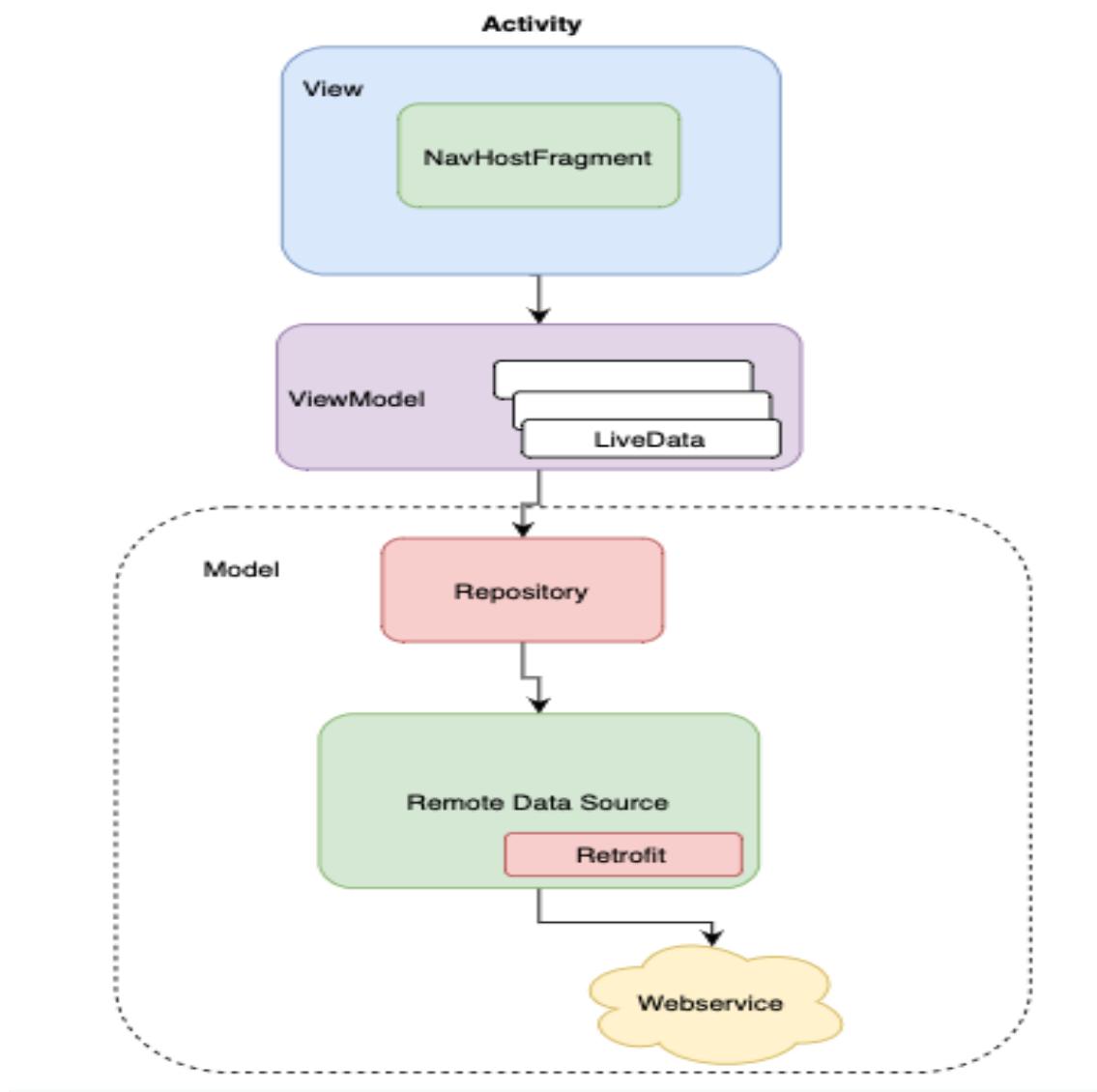


Figura 4.1: Diagrama de arquitectura general de la aplicación móvil.

A continuación, se describen a grandes rasgos cada componente de esta arquitectura.

### **View**

Consiste en el código de interfaz de usuario: el fragmento y el diseño XML. Cuando se necesita información, o el usuario realiza una determinada acción, se envía esa petición al *ViewModel*, pero no se obtiene la respuesta directamente. Para obtener la respuesta, la vista debe suscribirse a los observables que el *ViewModel* le expone.

## **ViewModel**

Funciona como un puente entre la *View* y el *Model*. Un *ViewModel* proporciona los datos para un componente de UI específico, como un fragmento o una actividad, y también incluye lógica empresarial de manejo de datos para comunicarse con el modelo. *ViewModel* no conoce los componentes de UI, de manera que no se ve afectado por los cambios de configuración.

Básicamente, lo que hace es interactuar con el modelo y exponer el observable que puede ser observado por la *View*.

## **Model**

El *Model* representa los datos y la lógica de negocios en la aplicación Android. En nuestro caso, contiene: modelos, repositorios y fuentes de datos provenientes desde el *backend*

### **4.1.2. Comunicación entre *View* y *ViewModel***

Una vez tenemos por un lado la *View* y por el otro lado el *ViewModel* vamos a explicar como realizar la comunicación entre ambos componentes.

El *ViewModel* va a comunicarse con el *Model* para traer la data necesaria. Cuando esa data cambia, se le debe informar de alguna forma al fragmento, y en ese momento entra en juego un componente llamado *LiveData*.

## **Live Data**

*LiveData* es una clase que retiene datos observables. Otros componentes de nuestra aplicación pueden supervisar cambios en objetos que usan este titular sin crear rutas de dependencia explícitas y rígidas entre ellos.

Usando *LiveData*, cada uno de nuestros fragmentos pueden recibir notificaciones cuando se actualizan los datos. Para recibir esas actualizaciones, un determinado fragmento debe observar de ese *LiveData*, para que cuando se actualicen los datos enterarse, y de esa forma, poder actualizar la *UI*. Un ejemplo de esto, se puede observar en la siguiente figura, cuando se observan las categorías, para poder reflejar las mismas en un *RecyclerView* de la *UI*.

```
private fun observeCategories(catAdapter: CategoriesAdapter, view: View){  
    homeVM.fetchCategoriesList.observe(viewLifecycleOwner, Observer { response ->
```

Figura 4.2: Observando categorías desde *HomeFragrment*

Una ventaja que tiene el hecho de utilizar *LiveData* es que el mismo esta optimizado para los ciclos de vidas, lo que implica que las referencias se borran automáticamente cuando ya no son necesarias.

#### 4.1.3. Comunicación entre *ViewModel* y *Model*

Vamos a explicar la forma en que se comunican los componentes *ViewModel* con el *Model*.

Nuestra aplicación cuenta con un *backend* que proporciona una API de REST. En ese sentido, usamos la biblioteca *Retrofit* para acceder a nuestro *backend*.

En función de eso, tenemos diferentes *WebServices* que lo que hacen es comunicarse con los diferentes endpoints que expone la API REST, esto se puede observar en el paquete *endpoints* como se observa en la figura a continuación.

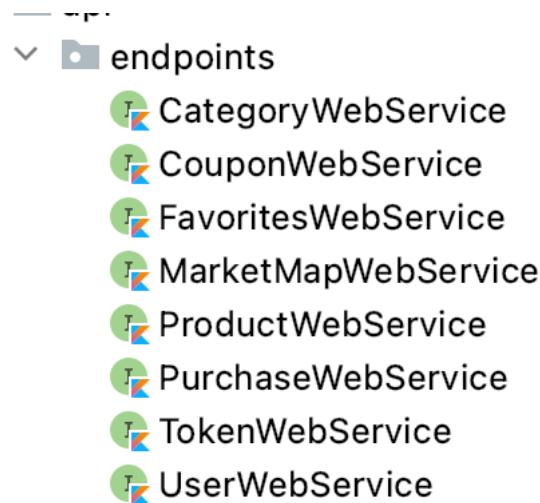


Figura 4.3: Paquete *endpoints* donde se establece comunicación la API expuesta en el backend.

Seguidamente, vimos como diseñar la comunicación entre los *WebServices* y los *ViewModels*. Una idea inicial para implementar *ViewModel* podría consistir en llamar directamente a *Webservice* a fin de recuperar los datos y asignarlos a nuestro objeto *LiveData*. Aunque este método funciona, el mantenimiento de nuestra aplicación se complica a medida que crece. Estaríamos asignando demasiada responsabilidad a los *ViewModel*, lo que implica que no estaríamos cumpliendo el principio de *separación de problemas*. Además, el alcance de un *ViewModel* está vinculado a un ciclo de vida de *Activity* o *Fragment*, lo que significa que los datos del *Webservice* se pierden cuando finaliza el ciclo de vida del objeto de UI asociado. Este comportamiento crea una experiencia del usuario no deseable.

En cambio, nuestro *ViewModel* delega el proceso de obtención de datos a un nuevo módulo, un repositorio.

Los módulos de repositorio manejan las operaciones de datos. Proporcionan una API limpia para que el resto de la app pueda recuperar estos datos fácilmente. Saben de dónde obtener los datos y qué llamadas de API deben hacer cuando se actualizan los datos.

Como se puede observar en la figura que se presenta a continuación de los paquetes y clases de la solución, se nota que tenemos un paquete *repositories.implementation*

y otro *repositories.interfaces*, es decir, que tenemos interfaces de los repositorios, y para cada uno de ellos implementaciones concretas, ya que de esta forma podemos lograr que el *ViewModel* dependa de una interfaz de repositorio y no del repositorio concreto, lo que implica que ante un cambio en el repositorio concreto, el *ViewModel* no se verá afectado debido a que está dependiendo de una abstracción y no de una implementación concreta. Es decir, los *ViewModel* no saben como se adquieren los datos, de forma que en un futuro podemos proporcionarle información de varias implementaciones de obtención de datos diferentes.

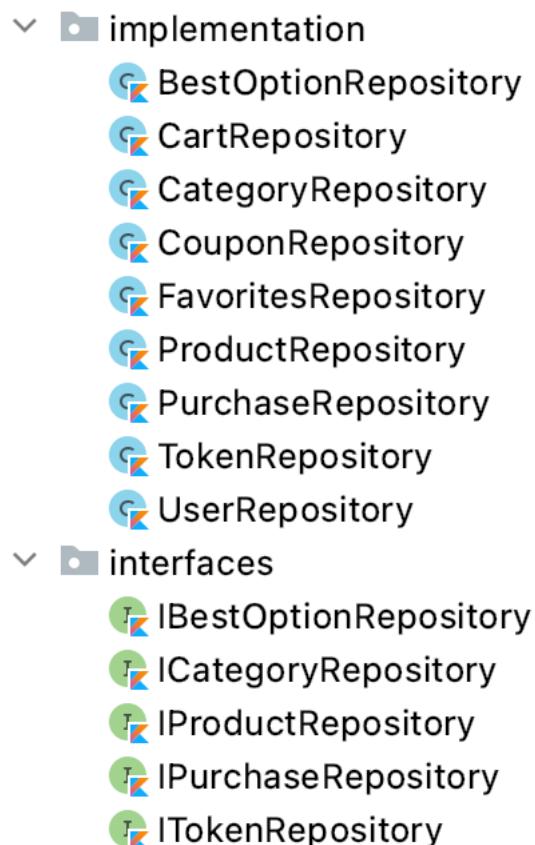


Figura 4.4: Paquetes `repositories.implementation` y `repositories.interfaces`

Cada una de las implementaciones de los repositorios, usan instancias de los *WebServices* para poder obtener los datos que se requieren, para eso decidimos utilizar inyección de dependencias de forma manual.

### Inyección de dependencias

Las implementaciones de repositorio, necesitan de una instancia de *WebService* para obtener los datos. Podríamos simplemente crearla, pero para hacerlo, también necesitaríamos conocer las dependencias de la clase *Webservice*. Esta situación nos obliga a duplicar el código, ya que cada clase que necesita una referencia a *WebService* debe saber cómo construirla con sus dependencias. Si cada clase crea un *WebService* nuevo, nuestra aplicación tendría un alto consumo de recursos.

Para solucionar este problema, usamos inyección de dependencias (en nuestro caso de forma manual), permitiendo así, que las clases definan sus dependencias sin construirlas. En el tiempo de ejecución, otra clase es responsable de proporcionar estas dependencias.

#### 4.1.4. Separación de problemas

Uno de los principios más importantes que se tomó fue el de la separación de problemas. Un error bastante usual que se comete al desarrollar aplicaciones es el de escribir todo el código en una *Activity* o *Fragment*.

Estas clases basadas en *UI* únicamente deberían contener lógica que se ocupe de interacciones del *SO* y de la *UI*. Al mantener estas clases tan limpias como sea posible, podemos evitar muchos problemas vinculados con el ciclo de vida.

Es de suma importancia considerar que nosotros como desarrolladores no somos propietarios de las implementaciones de *Activity* y *Fragment*, sino que ellas son clases que representan el contrato entre el *SO Android* y nuestra *app*. En función de esto, el S.O puede finalizarlas en cualquier momento, por ejemplo, por tener memoria insuficiente. En ese sentido, nosotros lo que intentamos hacer es reducir la dependencia de estas clases, teniendo únicamente en ellas lo que es la interacción entre el S.O y la UI para poder brindar una experiencia de usuario satisfactoria y mantenible en el tiempo.

#### 4.1.5. Single Activity Architecture

Las actividades son componentes a nivel del sistema que facilitan una interacción gráfica entre tu app y Android. La clase de actividad también permite que nuestra app reaccione a los cambios de Android, como cuando la IU de nuestra app ingresa al primer plano o sale de él, rota, etc.

Si bien las actividades son los puntos de entrada proporcionados por el sistema en la interfaz de usuario de su aplicación, su inflexibilidad cuando se trata de compartir datos entre sí y transiciones las ha convertido en una arquitectura que no se recomienda para construir nuestra aplicación.

Dentro del contexto de nuestra app, las actividades deben servir como un host para la navegación y deben contener la lógica y el conocimiento de cómo hacer la transición entre pantallas, pasar datos, etc. Sin embargo, siguiendo esta arquitectura, consideramos administrar los detalles de la UI en una parte más pequeña y reutilizable, como son los fragmentos.

Como se puede apreciar en la figura, se tiene una *activity* la cual contiene un *host* de navegación, que es un contenedor vacío en el que se intercambian los destinos a medida que un usuario navega por la aplicación.

De esa forma, la main activity se asocia con un gráfico de navegación, y contiene un *NavHostFragment* que es responsable de intercambiar los destinos según sea necesario.

En nuestro caso, tenemos una actividad principal con un *NavHostFragment* que se intercambia con los fragmentos de:

- **navhome:** Representa la pantalla de inicio de la aplicación, conteniendo los productos, categorías y ofertas.
- **navmap:** Contiene un mapa con los supermercados disponibles para el conjunto de productos disponibles, diferenciando la mejor opción con un marcador diferente en el mapa para captar la atención del usuario.
- **searchproduct:** Este fragmento, contiene una barra buscadora, las categorías de productos, y un *spinner* con criterios que permite ordenar la búsqueda realizada.
- **navfavorites:** Este fragmento, contiene la lista de productos favoritos del usuario permitiendo aquí también incorporarlos al carrito de compras.
- **navcart:** Este fragmento representa el carrito de compras, permitiendo ver los productos añadidos, aumentar o disminuir la cantidad que se desea adquirir de cada uno), y poder ir al fragmento *navmap* para visualizar los supermercados disponibles.
- **navsignup:** Este fragmento permite que un usuario de la aplicación se registre en la misma.
- **navlogin:** Este fragmento permite al usuario autenticarse en la aplicación, para desarrollar las funcionalidades que requieren autenticación.
- **navcoupons:** Este fragmento contiene los cupones que el usuario ha ido generando al utilizar la aplicación.

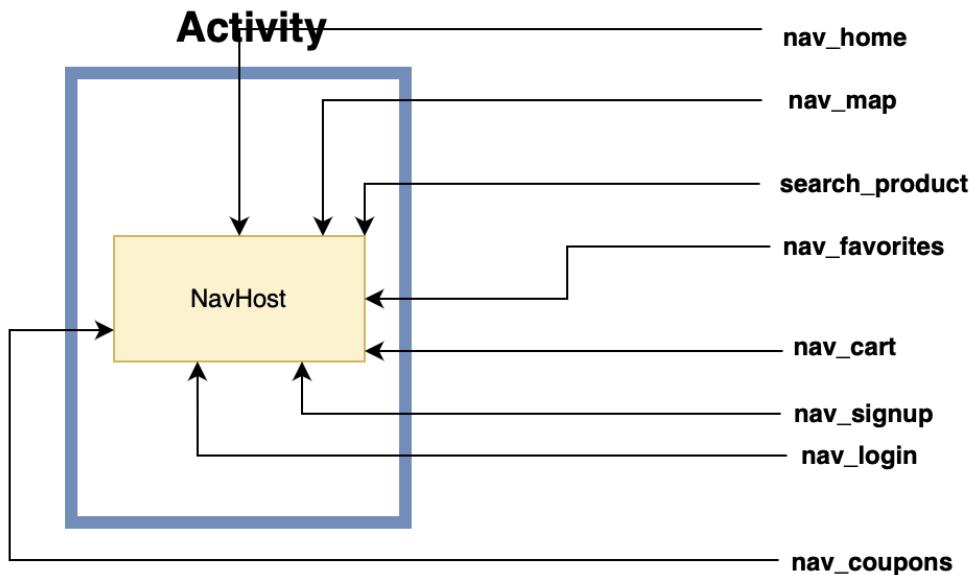


Figura 4.5: Aplicación de la arquitectura en aplicación AhorroUY.

#### 4.1.6. Navegación

Como se mencionó, nuestra aplicación cuenta de una única *activity* que contiene un *NavHostFragment* que se va intercambiando con los diferentes fragmentos a medida que el usuario va navegando por la aplicación.

La navegación ocurre entre los destinos de la aplicación, es decir, en cualquier lugar de la aplicación en el que los usuarios pueden navegar. Estos destinos están conectados a través de acciones.

Para diseñar las navegaciones entre los diferentes fragmentos de nuestra aplicación, utilizamos el gráfico de navegación, en donde el mismo contiene todos los destinos y acciones, representando todas las rutas de navegación de la aplicación.

El grafo de navegación de la aplicación, se puede observar en la figura que se presenta a continuación.

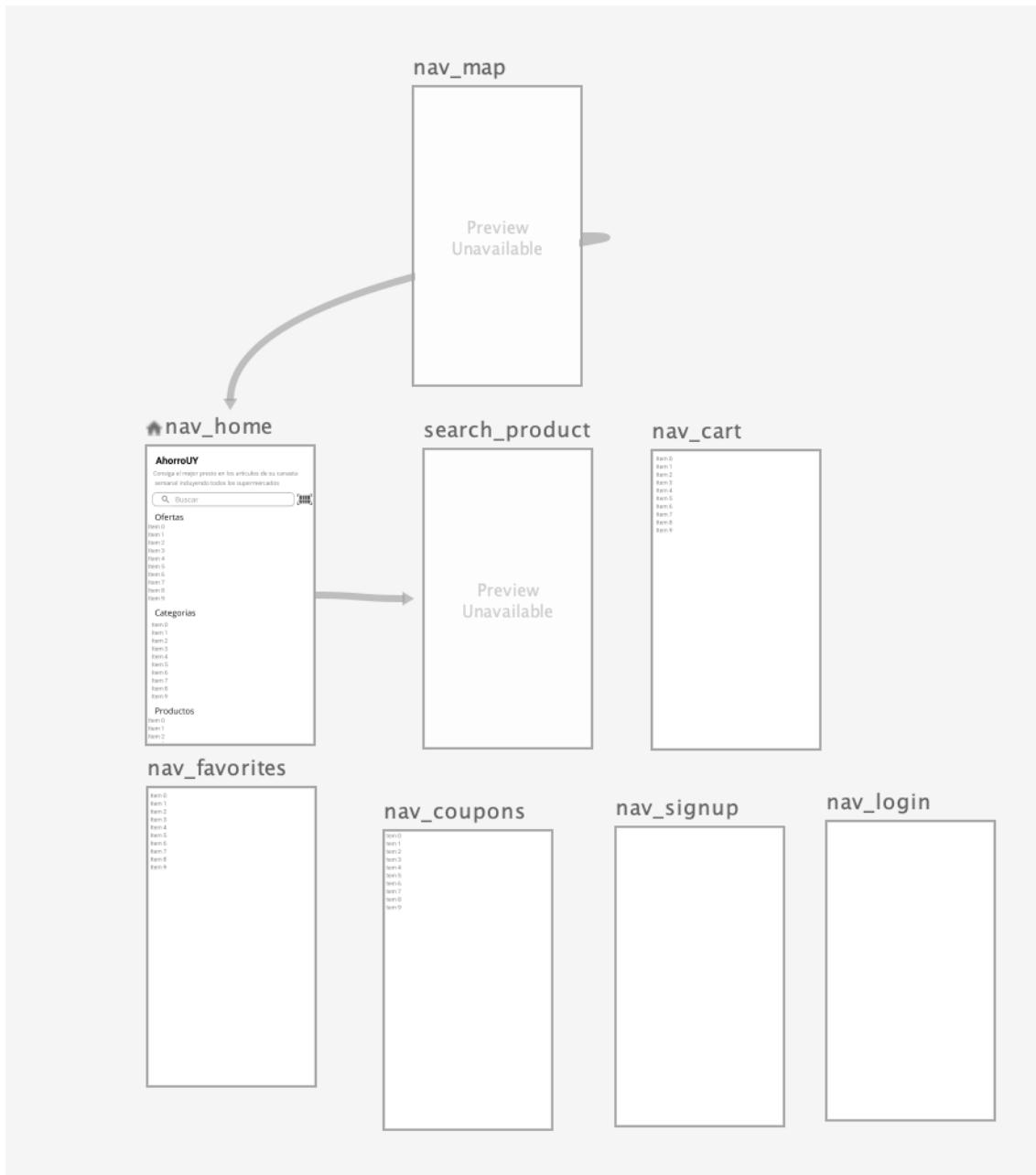


Figura 4.6: *NavigationGraph* de la aplicación AhorroUY.

#### 4.1.7. Librerías utilizadas

##### **KTX**

Android KTX es un conjunto de extensiones de Kotlin que se incluyen con Android Jetpack y otras bibliotecas de Android. Las extensiones KTX proporcionan Kotlin conciso a Jetpack, la plataforma de Android y otras API.

- implementation 'androidx.core:core-ktx:1.3.2'
- implementation 'androidx.appcompat:appcompat:1.2.0'

## ***Navigation***

El componente Navigation de Android Jetpack te permite implementar la navegación, garantizando una experiencia del usuario coherente y predecible, ya que se adhiere a un sistema establecido de conjunto de principios.

Para eso, se incluyeron las dependencias que se indican a continuación.

- implementation 'androidx.navigation:navigation-fragment-ktx:nav-version-alpha07'
- implementation 'androidx.navigation:navigation-ui-ktx:nav-version'
- implementation 'androidx.navigation:navigation-fragment-ktx:2.3.5'
- implementation 'androidx.navigation:navigation-ui-ktx:2.3.5'

## ***Constraint Layout***

Para crear diseños *responsive* y que los contenidos se adapten a los diferentes tamaños de pantalla, hicimos uso de *Constraint Layout*, incluyendo la dependencia que se indica a continuación.

- implementation 'androidx.constraintlayout:constraintlayout:2.0.4'

## **Material**

Material es un sistema de diseño creado por Google para ayudar a los equipos a crear experiencias digitales de alta calidad. Para hacer uso de ella, se incluyó la siguiente dependencia.

- implementation 'com.google.android.material:material:1.3.0'

## **Retrofit**

Retrofit es un cliente de servidores REST para Android, que permite hacer peticiones al servidor tipo: GET, POST, PUT, PATCH, DELETE y HEAD, y gestionar diferentes tipos de parámetros, parseando automáticamente la respuesta a un tipo de datos.

Para eso, requerimos de las siguientes dependencias.

- implementation 'com.squareup.retrofit2:retrofit:2.9.0'
- implementation 'com.squareup.retrofit2:converter-gson:2.6.0'
- implementation 'com.squareup.okhttp3:logging-interceptor:4.5.0'

## Manejo de imágenes

Para el procesamiento de las imágenes (categorías, productos, entre otros), hicimos uso de la librería *Picasso*. Además de esta, hicimos uso de una librería llamada *hdodenhof* para poder colocar imágenes circulares en una sección de la aplicación. Para hacer esto, requerimos las dependencias que se indican a continuación.

- implementation 'de.hdodenhof:circleimageview:3.1.0'
- implementation 'com.squareup.picasso:picasso:2.71828'

## Firebase

Para poder enviar notificaciones a los usuarios acerca de ofertas de los productos, hicimos uso de *Firebase*. Para eso, se agregaron las dependencias a *Firebase Cloud Messaging* y *Firebase Analytics*, como se puede ver a continuación.

- implementation 'com.google.firebaseio:firebase-messaging:22.0.0'
- implementation platform('com.google.firebaseio:firebase-bom:27.1.0')
- implementation 'com.google.firebaseio:firebase-messaging-ktx'
- implementation 'com.google.firebaseio:firebase-analytics-ktx'
- implementation 'com.android.support:multidex:1.0.3'

## Escáner código de barras

Para proveer la funcionalidad de búsqueda de producto mediante escaneo de código de barras, hicimos uso de la librería *ZXing Android Embedded*, agregando la dependencia que se indica a continuación.

- implementation 'com.journeyapps:zxing-android-embedded:4.2.0'

## Generación de códigos QR

Para la generación de códigos QR, hicimos uso de la librería *ZXing Core*, agregando la dependencia que se indica a continuación.

- implementation 'com.google.zxing:core:3.4.0'

## Mapas

Para hacer uso de los mapas de *Google* y poder acceder a la ubicación del usuario, se agregaron las dependencias que se indican a continuación.

- implementation 'com.google.android.gms:play-services-maps:17.0.1'
- implementation 'com.google.android.gms:play-services-location:18.0.0'

## Permisos

Para gestionar los permisos del usuario, hicimos uso de la librería *EasyPermissions*, ya que es un *wrapper* que nos simplifica la lógica básica de permisos del sistema.

Para hacer uso de la misma, se agregó la dependencia que se indica a continuación.

- implementation 'com.vmadalin:easypermissions-ktx:1.0.0'

## 4.2. UX

### 4.2.1. *Navigation Drawer*

Nuestra aplicación cuenta con un *Navigation Drawer* que permite acceder a los destinos y funcionalidades de la aplicación.

Decidimos usar un *navigation drawer* debido a que la aplicación cuenta con demasiados destinos (5) como para realizar una *bottom navigation*, justamente cuando se tiene cinco o mas destinos, *Material Design* recomienda usar un *navigation drawer* en lugar de una *bottom navigation*.

Este *navigation drawer* permite a los usuarios una navegación rápida a todos los destinos de la aplicación.

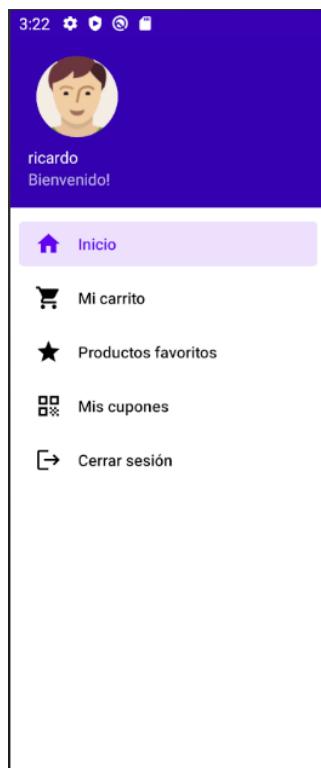


Figura 4.7: *Navigation drawer* presente en la aplicación.

#### 4.2.2. Mensajes de error

Para mostrar los mensajes de error al usuario como pueden ser: credenciales incorrectas, no hay productos disponibles para las búsquedas, entre otros, decidimos usar *Snackbars*, generalmente de baja prioridad, en donde los mismos desaparecen automáticamente sin la necesidad de que el usuario tenga que realizar acciones. Esto permite, que el usuario pueda ver fácilmente los mensajes de error y saber lo que está ocurriendo en el sistema.

#### 4.2.3. Carrito y productos favoritos

Un aspecto positivo a destacar en la aplicación, es que en la mayoría de los sitios de la aplicación se observa la opción de agregar al carrito un producto, así como agregar/quitar un producto de la lista de favoritos, visualizar el carrito actual de productos y poder buscar los supermercados disponibles para ese carrito. De esta forma, se contribuye a que el usuario tenga la facilidad de poder hacer uso de las funcionalidades principales del sistema en el lugar que se encuentre dentro de la misma, reduciendo de esta forma, la cantidad de interacciones que se deben realizar.



Figura 4.8: Sección Mis Favoritos de AhorroUY.

#### 4.2.4. Home page

En las siguientes imágenes, se puede observar la *Home Page* de la aplicación. Un aspecto que aporta a lo que es la *UX*, es que en la pantalla principal, el usuario es capaz de realizar las funcionalidades *core* del sistema como son: agregar productos al carrito, visualizar el mismo, buscar un producto por un determinado texto, buscar un producto escaneando mediante código de barras, y filtrar productos por determinadas categorías, lo que significa que el usuario puede hacer un uso básico de la aplicación sin la necesidad de andar navegando por las diferentes secciones de la misma.

Otro aspecto a destacar, es que decidimos colocar en la parte superior a las que se le denominan ofertas: aquellos productos cuyo precio actual es menor al precio regular, con un color que lo diferencia del resto de los productos, destacando su porcentaje de descuento, para de esta forma, poder captar la atención del usuario que hace uso de la aplicación.

Luego, se pueden observar lo que son las categorías de productos disponibles dentro de un *RecyclerView*, con iconos lo suficientemente claros para que el usuario pueda detectar de manera rápida cada una de las categorías.

Por último, se pueden observar el resto de los productos con un fondo blanco, debido a que como se mencionó anteriormente, se busca captar la atención del usuario sobre aquellos que se encuentran en oferta.

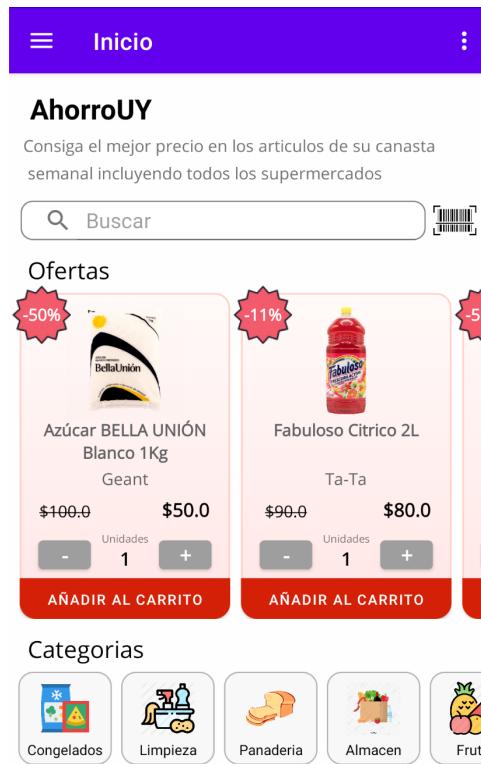


Figura 4.9: *Home page* de AhorroUY.

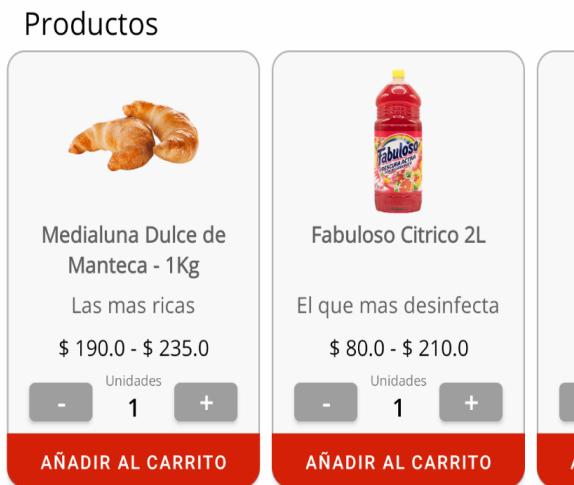


Figura 4.10: *Home page* de AhorroUY.

#### 4.2.5. Sección mapa con supermercados

Cuando el usuario presiona sobre el botón 'Buscar supermercados', se despliega un fragmento como el que se muestra en la figura a continuación.

Como se puede observar, se busca que el usuario pueda ver rápidamente la ubicación del supermercado por medio de un mapa (accediendo a su ubicación). Se puede notar que la opción recomendada, aparece marcada en el mapa con un ícono diferente (estrella roja) al resto de las opciones disponibles (comercio azul), con el fin de que el usuario pueda identificar rápidamente cual es el supermercado al que le conviene acudir a realizar la compra.

Por otro lado, consideramos que solo con el mapa no es suficiente para que el usuario pueda visualizar toda la información necesaria de los supermercados (sería incomodo tener todos los datos en el marcador), y debido a eso, decidimos colocar un componente *ListView* debajo del mapa con todas las opciones disponibles para realizar la compra de los productos. Se puede notar dentro de ese *ListView*, como la opción recomendada aparece resaltada en color amarillo para captar la atención del usuario y que el mismo pueda identificar claramente cual es el establecimiento que ofrece el precio más conveniente.

Dentro de ese componente, se le muestra además al usuario: el nombre del comercio, precio total para los productos, dirección y tiempo restante para la hora de cierre, todos aspectos fundamentales a la hora de decidir una compra.



Figura 4.11: Sección búsqueda de supermercados de AhorroUY.

## 4.3. Diseño de la arquitectura del Backend

### 4.3.1. Diagrama de descomposición de los *namespaces*

En las siguientes figuras se muestran los diagramas de descomposición de los *namespaces* del proyecto (utilizando el conector *nesting*). Por cuestión de visibilidad se decidió dividir el diagrama en dos (aunque realmente es uno solo que comprende todo el sistema).

Como se puede observar la solución está compuesta por los paquetes: *BusinessLogic*, *BusinessLogicInterface*, *BusinessLogicTest*, *BusinessLogicException*, *BusinessLogicForPushNotification*, *BusinessLogicForPushNotificationInterface*, *DataAccess*, *DataAccessInterface*, *DataAccessTest*, *Domain*, *DomainException*, *EntitiesForPushNotification*, *WebApi*, *WebApiModels* y *WebApiTest*.

Este primer diagrama mostrado tiene como objetivo mostrar la organización y jerarquía de paquetes (sin mostrar las dependencias, que se van a mostrar con otro diagrama que se presentará más adelante en el documento).

Se puede observar dentro de cada paquete, las entidades e interfaces que lo conforman. Podemos ver que en estos diagramas se utiliza el conector *nesting*, el cual es una notación gráfica para expresar la contención o anidamiento de elementos dentro de otros elementos. En este caso, lo utilizamos para mostrar de forma apropiada el anidamiento de paquetes en el diagrama de paquetes de nuestra solución.

En la figura (1) del diagrama de descomposición que se muestra a continuación, no se puede apreciar ningún conector *nesting* debido a que en la parte que se capturó no hay ningún paquete que se encuentre anidado a otro.

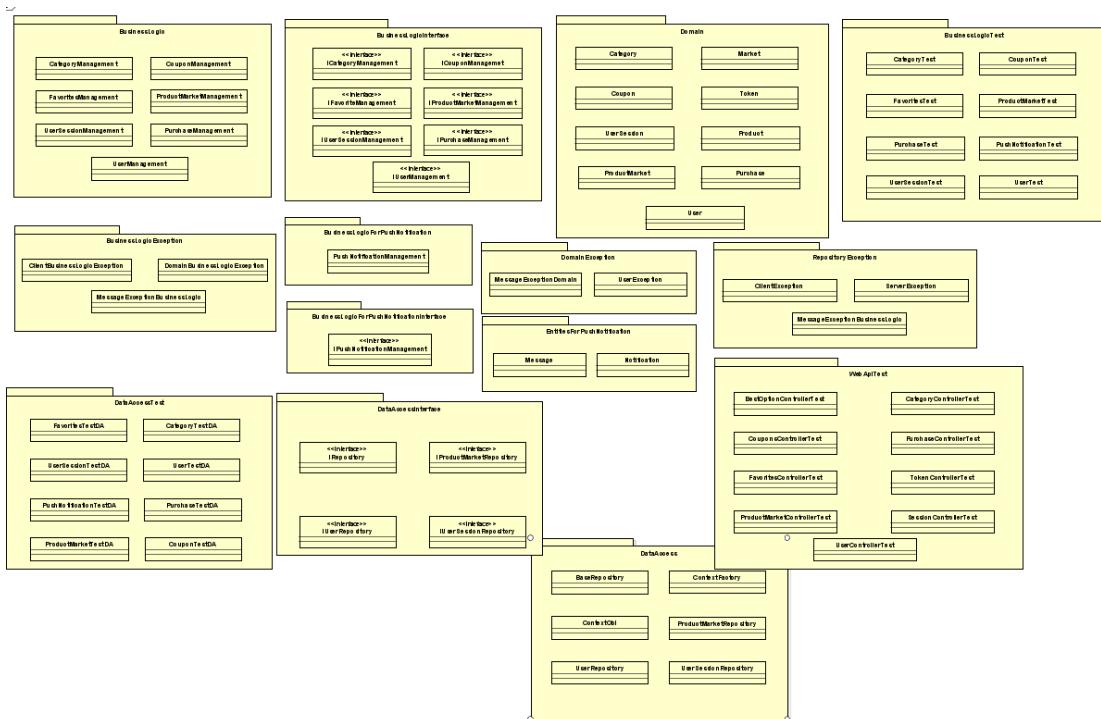


Figura 4.12: Diagrama sin conectores nesting

Sin embargo, en la figura (2), se pueden apreciar conectores *nesting* ya que tenemos presencia de paquetes anidados, podemos apreciar como dentro del paquete *WebApi* tenemos a los paquetes: *Filters* y *Controllers*.

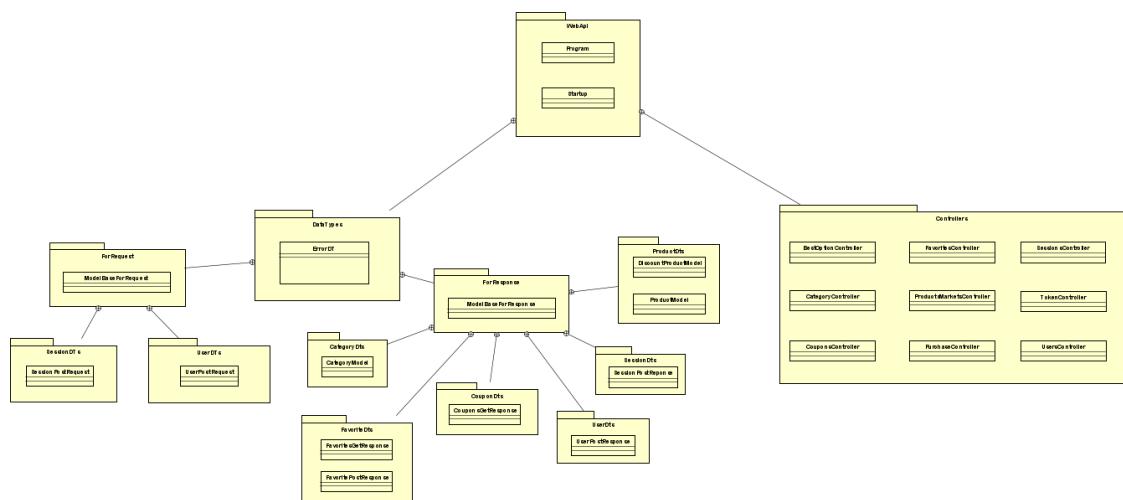
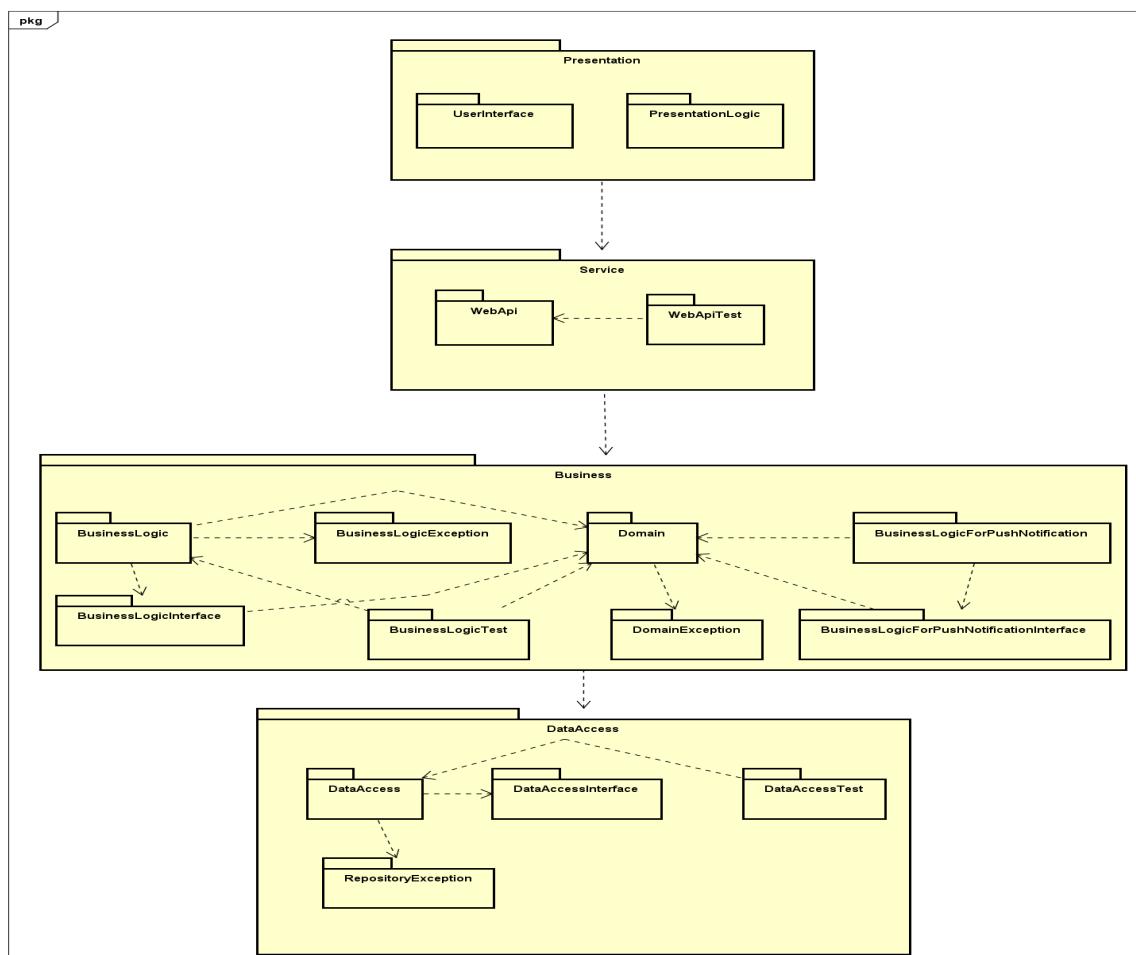


Figura 4.13: Diagrama con conectores nesting

Asimismo, se puede apreciar que dentro del paquete *Model*, tenemos los paquetes: *ForRequest* y *ForResponse*. El propósito de los mismos se explicará en detalle en una sección más adelante en este documento.

### 4.3.2. Diagrama general de paquetes (*namespaces*)



Como se observa en el diagrama de paquetes precedente, nuestra solución está compuesta por cuatro capas (horizontales), en donde cada una de ellas cumple un papel específico dentro de la aplicación.

Esta división de la arquitectura en capas trae grandes ventajas como la separación de responsabilidad entre los componentes. Donde los componentes dentro de una capa específica tratan solo con la lógica que pertenece a esa capa.

En nuestro caso la capa *Service*, esta compuesta por el paquete *WebApi* que contiene el paquete *Controllers*, *Models* y *Filters*, y su responsabilidad es la de manejar las solicitudes HTTP entrantes y enviar la respuesta a quien llama (en este caso la respuesta será hacia la capa de presentación).

La capa *Business* es la responsable de ejecutar las reglas de negocio específicas asociadas con la solicitud.

La capa *DataAccess* es la responsable de manejar todo lo relativo al almacenamiento de los datos.

La capa *Presentation* la cual es la responsable de manejar todo lo relativo a la interfaz de usuario (aplicación móvil) y realizar solicitudes a la capa de servicios. En particular, contiene los componentes de interfaz de usuario lanzados en esta versión, construidos en *Android*. Podemos notar que esta capa está compuesta por dos paquetes, uno de ellos es la interfaz de usuario en sí, mientras que el otro paquete existente es la lógica de la capa de presentación que son cada uno de los componentes existentes dentro de la interfaz de usuario, con la respectiva lógica que estos llevan.

#### 4.3.3. Responsabilidades de cada paquete

Centrándonos en las responsabilidades que tiene cada uno de los paquetes, comenzaremos por el paquete *WebApi*. Dentro de este paquete tenemos al paquete llamado *Controllers*, cuya responsabilidad es la de manejar las solicitudes HTTP entrantes y enviar la respuesta a quien llama. Este paquete, depende del paquete *BusinessLogicInterface* y de *BusinessLogicForPushNotificationInterface*, ya que debe conocer los servicios que ofrece las reglas de negocio para poder manejar las solicitudes entrantes y poder brindar una respuesta.

También, dentro de *WebApi* tenemos el paquete *Models*, el cual contiene DTOs (*Data Transfer Object*). El patrón DTO tiene como finalidad de crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple.

Por último, dentro de este paquete, tenemos al paquete *Filters*, los cuales nos permiten ejecutar código antes o después de determinadas fases en el procesamiento de una solicitud HTTP. Es decir, nos permite interferir una determinada solicitud antes o después de que llegó a nuestro *Controller*. En el caso de nuestra solución, se utilizó el filtro de excepciones (*Exception Filter*), el cual nos permite tener un mayor control sobre las excepciones producidas en el sistema, siendo que aplican políticas globales a las excepciones no controladas que ocurren antes de que se haya escrito el cuerpo de la respuesta, lo cual nos permite un mayor control sobre la respuesta si produce una excepción.

En cuanto al paquete *BusinessLogicInterface*, el mismo tiene la responsabilidad de exponer los servicios de las reglas de negocio, y permitir que los *Controllers* de la *WebApi* puedan hacer uso de los mismos sin conocer la implementación particular de las reglas de negocio.

Mientras que el paquete *BusinessLogicForPushNotificationInterface*, el mismo tiene la responsabilidad de exponer los servicios para enviar PushNotification, permitiendo a que la *WebApi* pueda hacer uso de los mismos para enviar notificaciones a los usuarios, sin conocer la implementación particular de las reglas de negocio. En este caso se tomó la decisión de no incluirlo dentro del paquete *BusinessLogicIn-*

terface debido a que no tiene la misma responsabilidades que los paquetes que se encuentran allí, esto debido a que son notificaciones que se envían a los usuarios.

Luego, tenemos el paquete *BusinessLogic*, que básicamente esta compuesto por clases que implementan las interfaces expuestas en el paquete *BusinessLogicInterface*, contiene la implementación de las reglas de negocio de nuestro sistema. A su vez, este paquete, depende del paquete *Domain* ya que debemos conocer las entidades del dominio de nuestro sistema para poder implementar las reglas de negocio, así como también de *BusinessLogicException*, ya que por medio del mismo es posible lanzar excepciones y detener la ejecución del sistema ante un comportamiento inesperado.

Continuando, también tenemos el paquete *BusinessLogicForPushNotification*, el cual está compuesto por clases que implementan las interfaces expuestas por el paquete *BusinessLogicForPushNotificationInterface*, conteniendo la implementación de las reglas de negocio acerca de las notificaciones a los usuario (las cuales son enviadas cada cierto tiempo). A su vez, este paquete, depende del paquete *Domain*, debido a que debe guardar una entidad del dominio, la cual es token del teléfono del usuario (esto para mandar a posteriori las notificaciones a dicho *token*) y también del paquete *EntitiesForPushNotification*, ya que estas son las entidades utilizadas para poder mandar la notificación, las cuales tienen los atributos necesarios para poder formar una notificación de forma correcta, donde dichas clases no pertenecen al dominio.

Podemos notar además, que el paquete *BusinessLogic* y *BusinessLogicForPushNotification* depende del paquete *IDataAccessInterface*, esta dependencia se basa en que es necesario que cada clase de las reglas de negocio contengan un objeto de la persistencia, que permita guardar y obtener objetos en el almacenamiento persistente. En el caso de *BusinessLogicForPushNotification* tiene dicha dependencia debido a que el propósito de nuestra aplicación es enviarle notificaciones cada cierto X tiempo a todas las personas que hayan descargado la aplicación y la hayan utilizado al menos una vez.

Luego, tenemos que el paquete *DataAccess* depende del paquete *IDataAccessInterface*, debido a que en *DataAccess* hay clases que requieren de las interfaces definidas en *IDataAccessInterface*.

Entonces, tenemos que, un modelo de alto nivel (*BusinessLogic*) no depende de un módulo de bajo nivel (*DataAccess*), sino que ambos dependen de una abstracción bien definida como *DataAccessInterface*. También encontramos que el modelo de alto nivel *BusinessLogicForPushNotification* no depende de un modulo de bajo nivel, sino que al igual que BL, ambos dependen de una abstracción bien definida como *DataAccessInterface*. Esta forma de implementación nos hace concluir que nuestra solución cumple con DIP (*Dependency inversion principle*).

Esta implementación tiene algunas ventajas que impactan en la mantenibilidad del sistema, ya que si el día de mañana, cambia la forma en que se almacena la

información, simplemente habría que agregar un nuevo objeto que implemente la interfaz *IRepository* (y por tanto las operaciones), pero no sería necesario realizar modificaciones a nivel de las clases de las reglas de negocio, ya que estas únicamente dependen de la abstracción, y no de implementaciones concretas.

Centrándonos en la responsabilidad del paquete *DataAccess*, el mismo se encarga del guardado de los datos del sistema. Es decir cómo se guardan y se obtienen los datos ya ingresados. Básicamente, una capa de persistencia encapsula el comportamiento necesario para mantener los objetos, es decir: leer, escribir y borrar objetos en el almacenamiento persistente.

Dentro del paquete *DataAccess* observamos que el mismo depende de las excepciones de repositorio (*Repository Exception*), esto debido a que cuando uno quiere acceder a datos que se encuentran alojados en algún almacenamiento persistente, se puede producir fallas a la hora de obtenerlos. De forma que debemos pensar que existe la posibilidad de fallas en el almacenamiento. A partir de esto, debemos diseñar una solución que esté preparada para afrontar estos flujos alternativos, y en particular, a considerar la manera en que en estos casos se debe cortar la ejecución del sistema, y mostrar al usuario un mensaje que indique el error que está ocurriendo. Es por esto, que el paquete *DataAccess* tiene una dependencia de las excepciones de repositorio (*Repository Exception*), es decir, sabe cuándo lanzar una excepción y qué mensaje mostrar gracias a la información que obtiene de *Repository Exception*. Estas excepciones a posteriori son capturadas inmediatamente por el filtro de excepciones.

En cuanto al paquete *Domain*, el mismo contiene las entidades que conforman el dominio del problema a resolver, y depende del paquete *DomainException* quien le permite lanzar excepciones generalmente cuando se desea crear una entidad del dominio que no cumple con las características mencionadas en el contrato (por ejemplo, cuando un usuario se registra y su nombre es vacío, esto produce una excepción la cual es capturada mostrándole un mensaje entendible al usuario).

#### 4.3.4. Mecanismo de acceso de datos

Para la implementación del acceso de datos se utilizó el *Repository Pattern*. El repositorio es una Fachada (Facade) que abstrae el dominio (o capa de lógica de negocio) de la persistencia. Se comporta como una colección (como un *IList* o un *ICollection*) escondiendo los detalles técnicos de la implementación.

Este patrón nos da la ventaja que nos permite utilizar las operaciones relacionadas a los datos, sin saber detalle alguno de la implementación, en nuestra aplicación los datos se persisten en una base de datos relacional, pero bien podrían ser almacenados en: un servicio externo, en archivos XML o simplemente en memoria, y para otros módulos de más alto nivel como la *BusinessLogic* sería totalmente indiferente, porque el mismo solo depende la abstracción y no de ninguna implementación concreta.

La forma en que decidimos implementar este patrón, es mediante un repositorio genérico (interfaz *IRepository*), el cual provee los servicios básicos de lectura y acceso a datos, conocidos como operaciones CRUD. Siendo estas *Create, Read, Update* y *Delete*. Lo que conlleva el uso de estas operaciones es resumir las funciones requeridas por un usuario para crear y gestionar datos. Además de esto las ventajas que introduce la utilización de las operaciones CRUD son la reunión en un solo elemento de configuración del software todas las acciones básicas que se realizan sobre una entidad de dominio. De igual manera mejora la reusabilidad del código, evitando la duplicación del mismo evitando generar diferentes implementaciones particulares de los repositorios por separado, lo cual conllevaría la repetición de código en abundancia, en las operaciones en común entre los repositorios específicos.

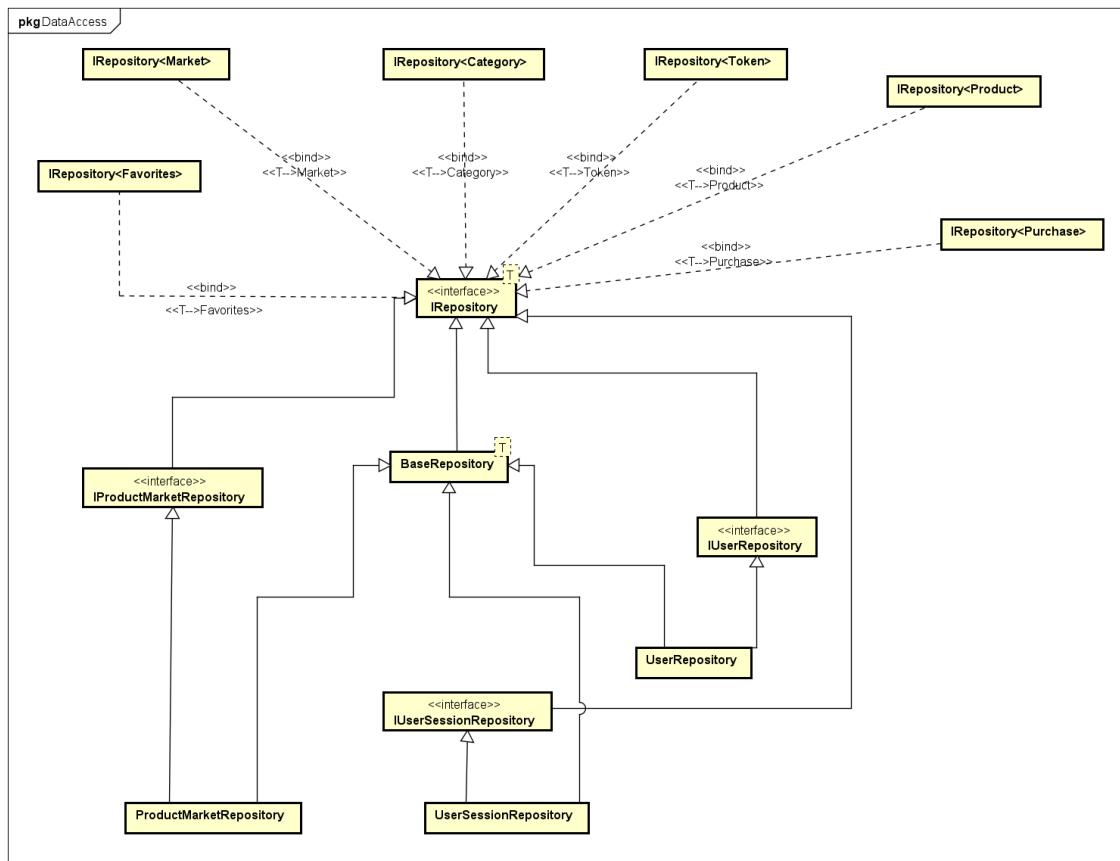
Luego, nuestro sistema cuenta con una clase genérica llamada *BaseRepository*, la cual se encarga de implementar las operaciones mencionadas previamente que se exponen en la interfaz *IRepository*, es decir, esta clase implementa de forma genérica la interfaz *IRepository*.

Pero con solo tener las operaciones CRUD no nos alcanza para desarrollar nuestra solución, en muchas ocasiones queremos obtener un determinado registro dada una condición, obtener registros por un determinado atributo que no sea el identificador, entre otros.

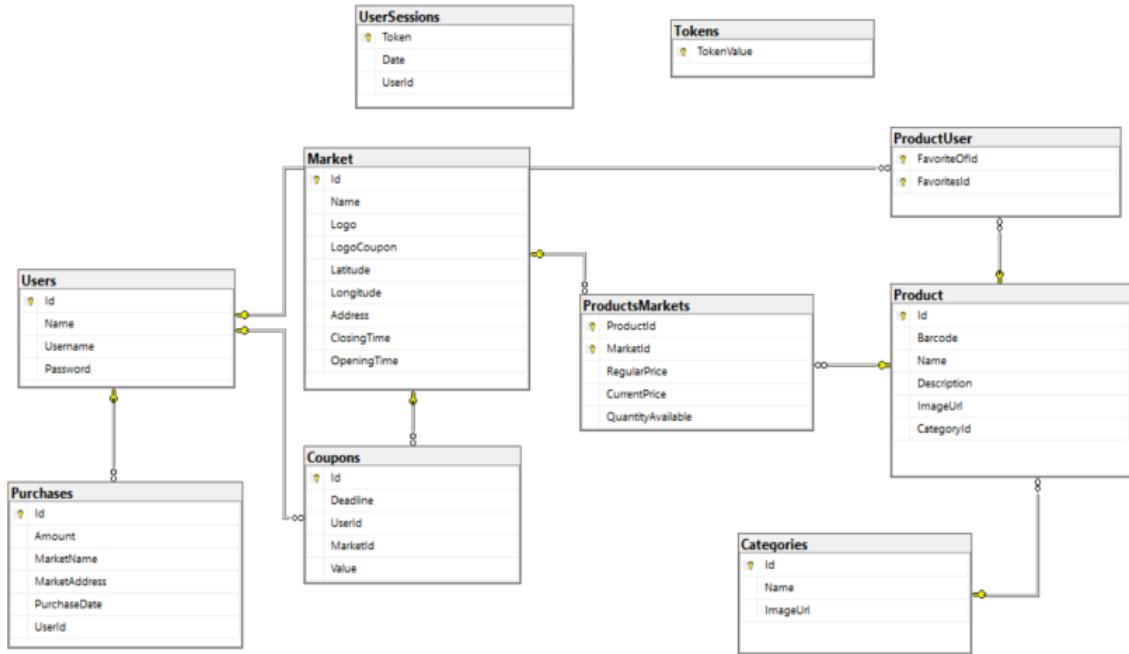
A raíz de esto, tenemos disponibles dos opciones para resolver ese tipo de solicitudes: la primera opción es la de solo manejar un repositorio genérico para cada entidad que se quiera persistir, y utilizar únicamente las operaciones CRUD. Esto significa que si queremos traer determinados registros por un determinado filtro, en primer lugar debemos hacer uso de la operación *getAll()* para obtener todos los registros de la base de datos, y luego en memoria filtrar sobre ese listado utilizando una determinada condición, lo cual no es muy bueno en términos de *performance*, ya que estamos trayendo todos los registros a memoria para luego recién aplicar el filtro correspondiente.

La otra opción disponible que fue por la que nos inclinamos es la de implementar para cada entidad que requiera de operaciones particulares, un repositorio particular con su correspondiente interfaz que exponga las operaciones del repositorio. Para cada entidad, lo diseñamos mediante una interfaz en donde se exponen las operaciones particulares que queremos que tenga el repositorio, esta interfaz hereda de la interfaz base genérica *IRepository* (donde se exponen las operaciones CRUD), y es implementada por una clase concreta que hereda de la clase llamada *BaseRepository* (clase en donde se implementan las operaciones CRUD). Como resultado de esto, tenemos para cada entidad que requiera operaciones particulares, una interfaz que expone los servicios de cada repositorio en concreto, además de poseer las operaciones CRUD, y por medio de esta, podemos desde las clases de las reglas de negocio hacer uso de las operaciones y evitar el filtrado en memoria tal como se mencionó previamente. Para observar las relaciones mencionadas previamente, se realizó el siguiente diagrama de clases (a alto nivel). No se incluyen los métodos contenidos

dentro de cada paquete, ya que lo que se desea mostrar es la relaciones de jerarquía de herencia e interfaces.



#### 4.3.5. Estructura de las tablas en la base de datos



La solución consta de 10 tablas, en donde cada una de ellas representa a una entidad del dominio, excepto las llamadas: *ProductUser* y *ProductsMarkets*, que representan las relaciones entre: Usuarios y Productos, y entre Productos e Mercados. Como la relación entre estas entidades es N a N, se crea la tabla antes mencionada, que nos permite obtener dado un usuario sus productos favoritos (y viceversa), así como también, dado un supermercado los productos que vende (y viceversa). La tabla *ProductsUser* al relacionar usuarios y productos posee dos claves foráneas: una hacia *Users* y la otra hacia *Products*. Mientras que la tabla *ProductsMarkets*, al relacionar productos con mercados donde se venden dichos productos, posee dos claves foráneas: una hacia *Markets* y la otra hacia *Products*.

Las restantes tablas representan a las entidades del dominio, es por esto que tenemos una tabla *Users*, que contiene todos los datos de los usuarios de la aplicación que se hayan registrado, y como clave primaria un ID.

Luego, la tabla *UserSessions*, mantiene almacenada la sesión de los usuarios mientras la misma esté activa, es decir, una vez que el usuario hace *logout*, se elimina su sesión de esta tabla.

La tabla *Purchases*, permite alojar toda la información relativa a las compras de un usuario. Al tener que guardar una compra de un cliente, posee una clave foránea hacia la tabla *Users*.

La tabla *Markets*, alberga toda la información relacionada con los supermercados disponibles en nuestro sistema. Básicamente la existencia de supermercados es clave dentro del sistema ya que sin ellos no se puede realizar ninguna funcionalidad.

La tabla *Coupons*, alberga todos los cupones existentes gracias a compras realizadas, por lo cual posee una clave foránea hacia la tabla *Markets* (para el supermercado que se emite el cupón) y la tabla *Users*, debido a que un cupón es para un usuario en específico (únicamente el usuario que obtuvo el cupón puede utilizarlo).

La tabla *Products*, alberga todos los productos existentes en el sistema. Son de igual importancia que los mercados, debido a que sin ningún mercado vende productos no se puede utilizar ninguna funcionalidad, por lo cual en dependencia de esto es necesario la existencia de la tabla '*ProductsMarkets*', ya que esto son los productos que se venden en un mercado. Posee una clave foránea hacia la tabla *Categories* debido a que un producto tiene una categoría específica.

La tabla *Categories*, permite alojar todas las categorías existentes para la clasificación de los productos. Esto permite dentro de la aplicación la búsqueda de productos por categorías.

Por último, la tabla *Tokens*, posee la información relativa a los tokens (valor del *token*) de los usuarios que hayan utilizado al menos una vez la aplicación.

### Mecanismo de borrado de elementos

Otro aspecto a destacar, es que decidimos realizar borrado físico en lugar de borrado lógico. Tomamos esta decisión, debido a que consideramos que es muy costoso mantener todos esos datos eliminados en la base de datos, que ya no tiene ningún sentido que permanezcan almacenados (no nos interesa mantener un historial de los registros de nuestra base de datos).

#### 4.3.6. Justificación del diseño para cada paquete

Dentro del paquete *WebApi*, tenemos el paquete *Controllers*, cuyo diagrama de clases se presenta a continuación. Este paquete está compuesto por las clases: *BestOptionController*, *CategoryController*, *CouponsController*, *FavoritesController*, *ProductsMarketsController*, *PurchaseController*, *SessionsController*, *TokenController* y *UsersController*. Cada una de estas clases se encarga de manejar las requests HTTP.

La clase *BestOptionController* posee la operación relativa a la obtención de las mejores opciones de compra, dependiendo de los productos seleccionados, la longitud y latitud del usuario así como la distancia máxima seleccionada. A partir de dichos datos, se procede a obtener los supermercados que tienen disponibles los productos seleccionados y se encuentran a igual o menor distancia que la máxima seleccionada.

La clase *CategoryController* posee la operación relativa a la obtención de las diferentes categorías de productos existentes. Esto permite, que dentro de la aplicación se pueda realizar la búsqueda de productos mediante categorías, ya que los productos dentro de la aplicación cuentan con una categoría específica.

La entidad *CouponsController* posee las operaciones relativas a los cupones de descuento de un usuario: permitiendo la obtención de todos los cupones asociados a un usuario, así como la generación de un nuevo cupón cuando el monto de una compra supera los \$1000 pesos uruguayos.

La clase *FavoritesController* posee las operaciones relativas a los productos favoritos de un usuario: permitiendo agregarlos, eliminarlos y obtenerlos.

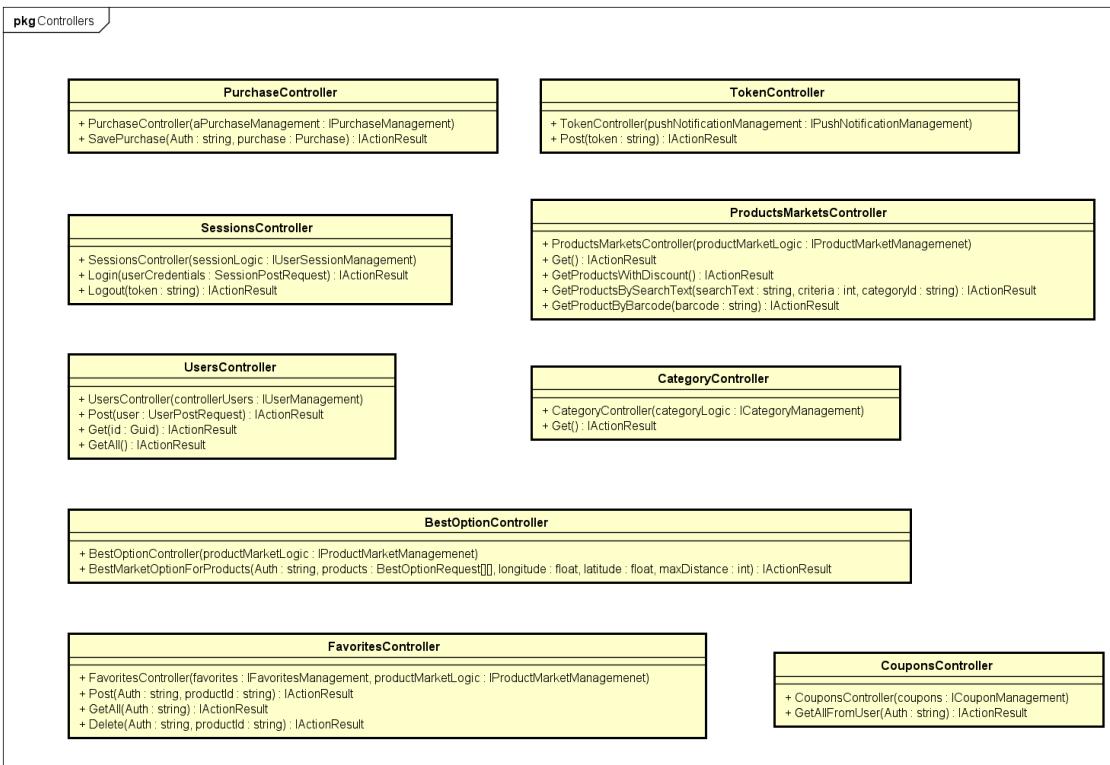
La clase *ProductMarketController* posee las operaciones relativas a los productos que se encuentran asociados a un supermercado: es decir, obtener los productos que se venden en un supermercado, obtención de los productos con descuentos que se venden en los supermercados, obtención de los productos buscados mediante texto, categoría o criterio que se encuentren disponibles en los supermercados y obtención del producto buscado mediante código de barra (esto debido a que el código de barra es un identificador único universal).

La clase *PurchaseController* posee las operaciones relativas al guardado de una compra realizada por un usuario.

Por otro lado, tenemos la clase *SessionController* que maneja todas las operaciones referidas a los inicios de sesión dentro de la aplicación: *login* y *logout*.

Sin embargo, tenemos la entidad *TokenController* que maneja la operación del agregado del *token* del teléfono de un usuario que utiliza la aplicación por primera vez.

Por último, tenemos el *UserController* que maneja todas las operaciones referidas a los usuarios del sistema, entre las que se encuentran las operaciones CRUD.



La idea detrás del diseño de una API es poder exponer una interfaz que sea amigable para los desarrolladores que se integran a ella y a su vez que sea lo suficientemente robusta para evitar que surjan errores en la interacción cliente-servidor o *frontend-backend*. Por ende, todas las decisiones que tomamos velan por cumplir estos objetivos. Los DTOs por lo general nos ayudan a esto, a que tengamos un control estricto sobre lo que “aceptamos nos pasen” (*requests*) y sobre lo que “queremos devolver” (*responses*).

Los DTO (*Data Transfer Object*) los utilizamos para diseñar la forma en que la información debe viajar desde la capa de servicios a la aplicación o capa de presentación, ya que si usáramos directamente las entidades del dominio para retornar los datos en las *requests* lo que ocurriría es que terminamos retornando más datos de los necesarios, y exponiendo información que realmente no deberíamos exponer. Por esta razón, decidimos utilizar modelos en lugar de retornar directamente las clases del dominio. Otra decisión de diseño que decidimos tomar en cuanto a los DTO, es tener modelos para las *requests*, llamémosle *ModelsForRequest* y otros modelos para las *responses* llamados *ModelsForResponse*.

De esta forma tenemos dos tipos de DTOs en la API: los que modelan *requests* (por ej: *UserPostRequest*) y los que modelan *responses* (por ej: *UserPostResponse*).

Esta decisión se basa principalmente en que en la mayoría de los casos, los datos que recibimos en una *request* para realizar una determinada operación, no son los mismos que queremos mostrar cuando nuestra *API* da respuesta ante un HTTP GET a un determinado *endpoint*. Un claro ejemplo de esto, es en el funcionamiento

del *login*, cuando se realiza un HTTP POST para loguearse, sería deseable tener un modelo que incluya el *email* y la *password*, sin embargo, para dar respuesta a un HTTP GET a un *endpoint* de usuarios no voy a querer retornar su *password* por obvias razones de seguridad. Vease esto en los siguientes diagramas.

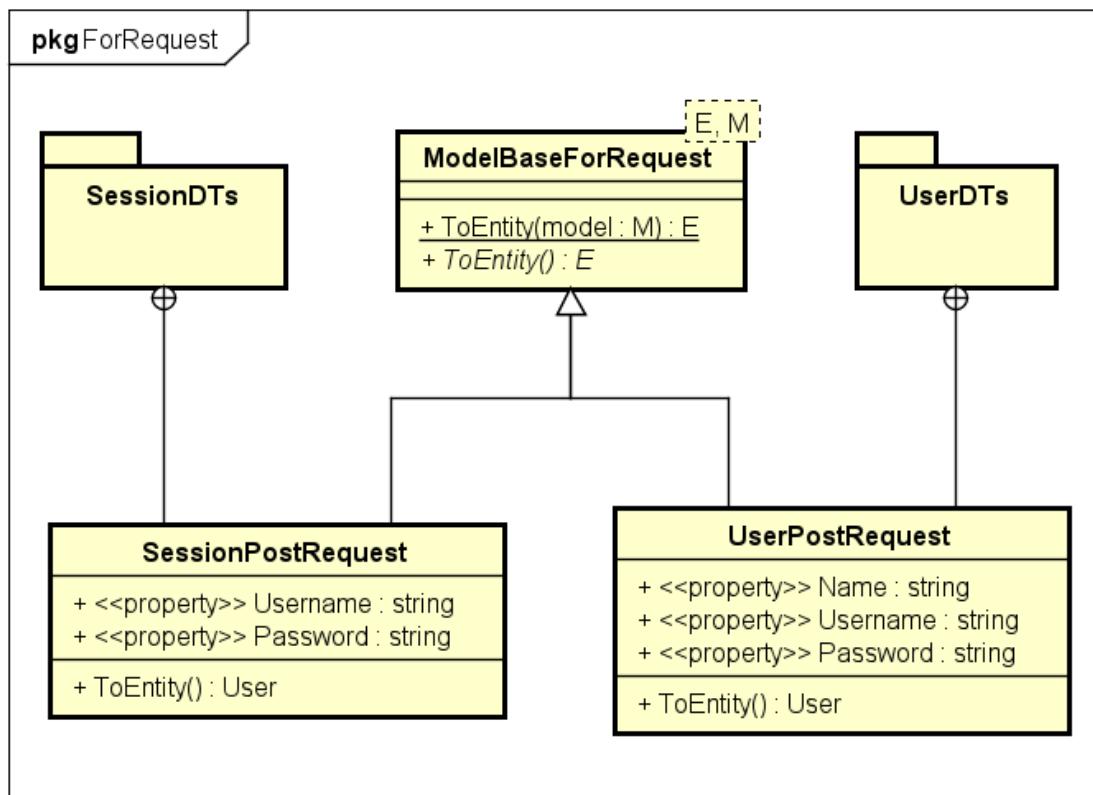


Figura 4.14: Diagrama de DTO's para la solicitud

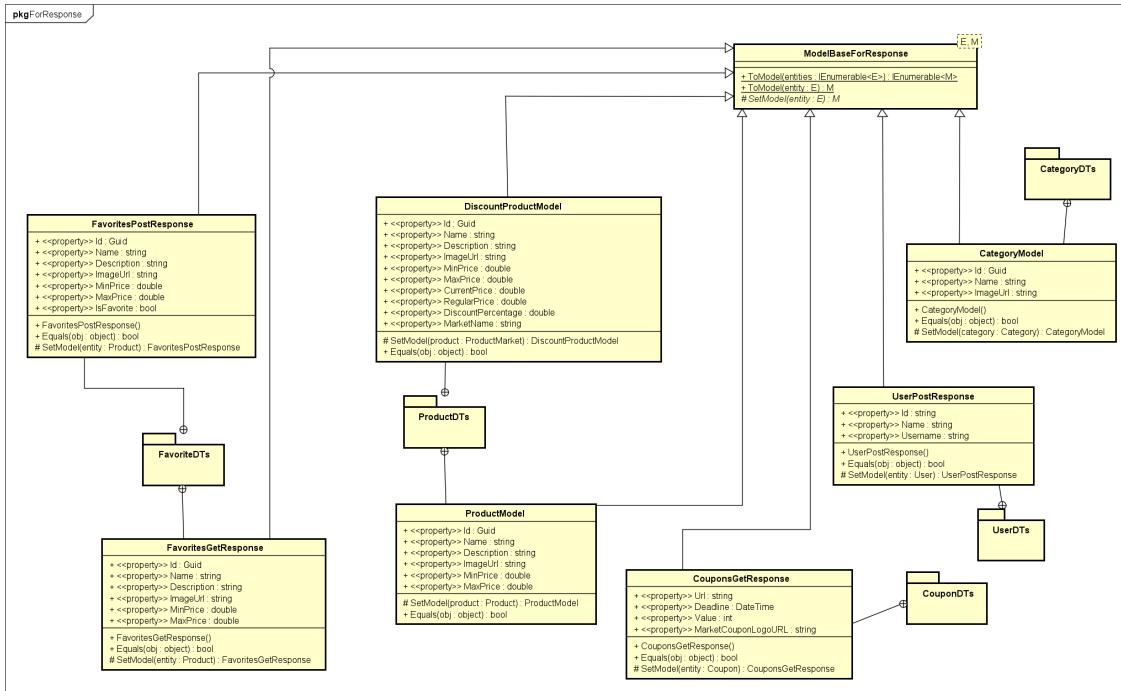


Figura 4.15: Diagrama de DTO's para la respuesta

Entonces justamente para que los controladores ya mencionados puedan resolver las solicitudes HTTP provenientes y puedan brindar una respuesta necesitan conocer los servicios ofrecidos por las reglas de negocio (*BusinessLogicInterface*).

Siendo así que cada uno de los *controllers*, poseen un objeto de la *BusinessLogicInterface* relativo a la entidad a la que se hace mención en su nombre (excluyendo de esto a los controladores que no representan entidades como tal). Este tipo de objeto, lo que nos permite es conectar a la *WebApi*, con los servicios brindados por la lógica de negocio, y hacer uso de las operaciones que ella presenta para poder resolver las solicitudes.

El hecho de que la *WebApi* depende de una abstracción de las reglas de negocio, y no de una implementación concreta, favorece a la mantenibilidad de la aplicación, ya que estamos cumpliendo con el **DIP (Dependency Inversion Principle)**, evitando que un modulo de alto nivel (*WebApi*) dependa de uno de bajo nivel (*BusinessLogic*), y haciendo que ambos dependan de abstracciones bien definidas. Entonces, si en el futuro cambia la forma en que están implementadas las reglas de negocio, los controladores de la *WebApi* no se verán afectados en lo absoluto porque están dependiendo de una abstracción y no de las implementaciones particulares.

Dado lo mencionado, es importante notar los nombres tanto de las interfaces de las reglas de negocio, como la implementación de las mismas se posee en su nombre el término '*Management*'. Haciendo énfasis en esto debemos especificar que una de las decisiones tomadas fue evitar el uso de una única clase conocida como '*System*' o '*Sistema*' en español. Por lo cual para cada una de las entidades que se encuentra

en el Dominio, se le realizo una interfaz del manejador que le permite gestionar los eventos pertinentes a esta entidad, con la correspondiente implementación. Tal como se menciona en el párrafo anterior, esto se realiza de esta forma, para evitar depender de las implementaciones específicas, y estar dependiendo únicamente de los servicios expuestos, de manera que si se cambia la implementación, los servicios no se vean afectados por dicho cambio. Ya que se dependerá únicamente de la exposición de los servicios, y no de cómo ellos están implementados. Esta decisión fue tomada, en lo que expone el autor Robert C. Martín en su libro *Clean Code*, en la sección de Responsabilidad Única del capítulo 10 (página 132-138). [9] Martín nos dice que una clase debe tener uno y solo un motivo para cambiar. ¿A qué nos referimos con esto? En particular sí se desarrollara una única clase Sistema, la cual tuviera múltiples persistencias y múltiples métodos públicos, según el principio que describió Martín, lo que sucedería es que dicha clase tendrá múltiples responsabilidades, describiendo dichas responsabilidades serían:

1. Se encargaría de verificar el formato de cada uno de los objetos que se crean.
2. Se encargaría de agregar a las diferentes tablas los diferentes tipos de objetos creados.
3. Se encargaría de eliminar a los objetos de las diferentes tablas.
4. Se encargaría de 'tirar' excepciones una vez verificado el formato.

Entonces la justificación en la que nos basamos para no utilizar una única clase la cual es responsable de manejar todo y también una única interfaz la cual es responsable de exponer los servicios brindados por las reglas de negocio, es que la implementación de dicha interfaz tendría múltiples responsabilidades ya que tiene diferentes tipos de objetos asociados. Es por esto que decidimos crear para cada una de las clases del dominio una interfaz de las reglas de negocio, y también una implementación de las mismas, en las cuales las únicas responsabilidades que tienen, es manejar los objetos del tipo del objeto del dominio y encargarse acerca de sus eventos.

Además el logro que conlleva dividir una clase sistema en diferentes subsistemas, lleva al cumplimiento del ***GRASP Controller***. Estos pequeños subsistemas, constituyen diferentes controladores, y cada uno de estos, cumplen con que la única responsabilidad que tienen es el manejo de los objetos del tipo del objeto del Dominio, los cuales están claramente identificados dentro del nombre del controlador (siendo así que en un comienzo le llamamos manejadores, pero el término correcto son controladores).

Otro punto importante a destacar del patrón, es que nos dice que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control, lo cual se cumple en su totalidad dentro del proyecto, en el cual el dominio está dividido en su totalidad de las reglas de negocio, donde se encuentra una dependencia en que las reglas de negocio dependen del dominio.

También en consecuencia de la subdivisión de diferentes controladores (no en sentido de los existentes en la *WebApi*), nos lleva al cumplimiento de dos patrones diferentes. Siendo estos, el patrón ***GRASP de Alta cohesión*** y también el patrón ***GRASP de Bajo acoplamiento***.

- Alta cohesión: Es la medida en la que un módulo de un sistema tiene una sola responsabilidad. Por ende, un módulo con alta cohesión será aquel que guarde una alta relación entre sus funcionalidades, manteniendo el enfoque a su único propósito. Nótese cómo este principio se relaciona de manera casi directa con el principio de responsabilidad única desarrollado anteriormente. Lo cual nos lleva a decir, que sí hubiera una única clase la cual se conociera como “*System*”, lo que sucedería es que dicha clase tendría múltiples responsabilidades como mencionamos anteriormente, pero además tendría una baja cohesión ya que no tendría una única responsabilidad, debido a que no tendría una alta relación entre sus funcionalidades. Lo que nos lleva a decir que cada uno de los “manejadores” tiene una alta cohesión, ya que la responsabilidad que tienen siempre está orientada al mismo tipo de objeto del dominio, es decir no tiene múltiples responsabilidades con objetos del dominio. Cumpliendo así el GRASP de alta cohesión, debido a la subdivisión en pequeños “*System's*”.
- Bajo acoplamiento: Siendo el acoplamiento la relación que se guardan entre los módulos de un sistema y la dependencia entre ellos. El bajo acoplamiento dentro de un sistema indica que los módulos no conocen o conocen muy poco del funcionamiento interno de otros módulos, evitando la fuerte dependencia entre ellos. Lo que también nos lleva a relacionarlo con el principio SOLID abierto/cerrado debido a que si no tenemos mucha dependencia entre los módulos, podemos extender el comportamiento de un módulo sin afectar a otro. Este bajo acoplamiento está dado por el hecho de que los módulos de alto nivel como es el caso de la *WebApi* no dependen de los de bajo nivel como las implementaciones de las reglas de negocio (*BusinessLogic*), sino que ambos dependen de interfaces bien definidas que exponen los servicios y reducen el acoplamiento entre los módulos. De forma análoga sucede con los módulos de *BusinessLogic*, ya que los mismos no dependen de las implementaciones específicas de los módulos de *DataAccess*, sino qué sucede que dependen de las abstracciones de los mismos, sin importar cómo estos están implementados. Lo cual conduce a poder decir que cada módulo de *BusinessLogic* tiene asociado una o varias abstracciones de la persistencia.

# 5. Especificación API

## 5.1. API REST

En particular primero debemos decir que es una API. Donde la misma, es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones. Una API permite que sus servicios se comuniquen con otros sin necesidad de saber cómo están implementados, otorgando flexibilidad, simplificación en el diseño, entre otras características.

Pero por otro lado, encontramos a REST. Siendo que es un estilo de arquitectura para la creación de sistemas distribuidos basados en hipermédia. REST es independiente de cualquier protocolo subyacente y no está necesariamente unido a HTTP. Sin embargo, en las implementaciones más comunes de REST se usa HTTP como protocolo de aplicación, y esta guía se centra en el diseño de API de REST para HTTP, tal como es el caso de nuestro sistema. REST se compone de una lista de principios que se deben cumplir en el diseño de la arquitectura de una API.

### 5.1.1. Principios REST

Pasaremos a analizar y verificar que nuestra API cumple con cada uno de los principios REST.

#### Interfaz uniforme

- Nuestra API está diseñada en torno a recursos, que son cualquier tipo de objeto, dato o servicio al que puede acceder el cliente. Por ejemplo, tenemos el recurso User (Id, Nombre, Apellido, Nombre de usuario, contraseña).
- En nuestra API, un recurso tiene un identificador, que es un URI que identifica de forma única a ese recurso.
- Los clientes interactúan con un servicio mediante el intercambio de representaciones de recursos. El servidor envía los datos vía JSON pero el mecanismo de almacenamiento interior (una base de datos en nuestro caso) para el cliente es transparente.
- La representación del recurso que le llega al cliente, es la suficiente para que el mismo pueda crearlo, modificarlo y borrarlo, esto suponiendo que para estas

acciones el mismo tenga permisos. Esto debido a que incluye el uso de verbos HTTP, como GET, POST, PUT y DELETE.

- Se utilizan mensajes descriptivos por medio de las características del protocolo HTTP, las cuales son verbos y códigos de estado.
- Para obtener una API jerárquica y con ciertas reglas, procuramos el uso de nombres en **plural**.

### Peticiones sin estado

Las API REST usan un modelo de solicitud sin estado. Las solicitudes HTTP deben ser independientes y pueden producirse en cualquier orden, por lo que no es factible conservar la información de estado transitoria entre solicitudes, es por esto que decimos que todas las consultas tienen la información necesaria para entenderla y no se pueden utilizar datos almacenados. El estado no guardará información de las últimas consultas y tratará a cada una como una nueva.

Por ejemplo tenemos que:

- GET api/users/12345
- DELETE api/users/12345

En la segunda petición hemos tenido que indicar el identificador del recurso que queremos borrar. El servidor no guarda los datos de la consulta previa que tenía el cliente en particular. Una petición del tipo DELETE api/users debe dar error, ya que falta el identificador y el servidor almacena los datos.

### Cacheable

Cuando se manda una request se puede establecer que se cachee o no, en caso de que se cachee, se le da la posibilidad al cliente de usar la respuesta después.

En nuestra API decidimos no cachear las respuestas del servidor, sabiendo de que no será muy alto el número de solicitudes hacia un determinado recurso, y por lo tanto, no tendremos problemas de rendimiento en ese sentido.

### Separación de cliente y servidor

Esto se refiere a que para que REST exista deben haber dos actores, un cliente que consuma y un servidor que almacene o genere información. En ese sentido, nuestra API sigue los siguientes criterios:

- El cliente y servidor están separados, su unión es mediante la interfaz uniforme.
- Mientras la interfaz no cambie, podremos cambiar el cliente o el servidor sin problemas.

## Sistema dividido en capas

Capas distribuidas jerárquicamente en la que se restringe el comportamiento de forma que disminuya la visibilidad directa entre componentes. Además, permite que se dividan las capas en distintos servidores.

En ese sentido, nuestra API sigue los siguientes criterios:

- Un cliente no sabe inicialmente (ni debería interesarle) si está conectada directamente con el servidor o si hay componentes intermedios que incrementan la seguridad o distribuyen la carga entre los servidores de la aplicación.
- El uso de capas intermedias, en un futuro servirá para aumentar la escalabilidad o para implementar nuevas políticas de seguridad.

## 5.2. Descripción general de códigos de error

Primero debemos introducirnos acerca de que es un código de estado HTTP. De forma tal que un código de estado HTTP es un mensaje que devuelve el servidor cada vez que el navegador realiza una petición al servidor. Si el servidor es capaz de devolver el contenido que solicita el navegador y no existe ningún error, estos códigos HTTP no son visibles para el usuario. En cambio, si algo va mal, el servidor devuelve un código de estado HTTP que indica que algo no salió como esperaba con un mensaje de error incluido en la implementación de nuestra WebApi, de forma tal que es entendible para el usuario de aplicación entender que fue lo que ha salido mal.

En función del código de estado que devuelve el servidor, el usuario de la WebApi, además del mensaje de error, observando el código de estado HTTP se podrá hacer una idea del tipo de error que se ha producido. De forma que es un buen principio utilizarlos cuando se tiene una API, ya que es lo que da noción de qué situación el usuario se encuentra cuando realiza peticiones a la misma.

Es por eso que encontramos que los códigos de estado los podemos dividir en cinco categorías diferentes, de manera que visualizando únicamente el primer dígito del código de estado le permite identificar al usuario de la API, al tipo de respuesta que la misma le está dando. Siendo así que las diferentes categorías son:

- Código de estado 1xx: Se trata de respuestas de carácter informativo donde el servidor le notifica al cliente que la petición actual continua.
- Código de estado 2xx: Se trata de respuestas que son satisfactorias. Indica que la acción solicitada por el cliente ha sido recibida, entendida, aceptada y procesada correctamente.
- Código de estado 3xx: Se trata de códigos los cuales hacen referencia a que la solicitud ha sido recibida. Sin embargo, para asegurar un procesamiento exitoso es necesario que el cliente tome una acción adicional como, por ejemplo, una redirección.

- Código de estado 4xx: Estos códigos de estado hacen referencia a errores a que se ha presentado un error de cliente. Esto quiere decir que se ha recibido la solicitud, pero esta no se puede llevar a cabo.
- Código de estado 5xx: Estos códigos HTTP también muestran errores, pero por el lado del servidor.

Sucede entonces que existen más de 70 códigos de estado HTTP, esto da la idea de que todos los códigos existentes se deberían utilizar lo cual no es cierto, ya que la mayoría de los desarrolladores no conocen a todos, de tal manera que si se utilizan códigos de estado poco conocidos, se está guiando a qué se consuma nuestra API, pero luego se redirigen a otra pagina a buscar cual es el significado del código de estado qué les está llegando.

De tal manera que siguiendo con las buenas prácticas en el desarrollo de la API, decidimos utilizar no mucha cantidad de códigos de estado, y tampoco códigos pocos conocidos.

Entonces, en resumen, todo lo anterior nos conlleva a preguntar, ¿cuántos códigos de estado debe usar en la API? Y la respuesta que encontramos a esto fue lo que se implementó de manera satisfactoria en nuestra API. Se podrían implementar con los códigos más utilizados (200, 400 y 500), y a partir de allí extenderlos dependiendo de las necesidades de la aplicación. Pero en particular la API no debería tener más de ocho códigos de error. Esto debido a que una buena práctica, es utilizar pocos y conocidos códigos de estado HTTP.

Siendo así que los códigos de estados HTTP que se utilizan y devuelven en nuestra API son:

Código de estado	Mensaje de estado	Descripción
200	Ok	La solicitud ha tenido éxito.
201	Creado	La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición POST.
204	Sin contenido	La petición se ha completado con éxito pero su respuesta no tiene ningún contenido, aunque los encabezados pueden ser útiles
400	Solicitud incorrecta	No se puede procesar la solicitud porque tiene un formato incorrecto o es incorrecta.
401	Sin autorizar	La información de autenticación necesaria falta o no es válida para el recurso.
403	Prohibido	Se denegó el acceso al recurso solicitado. Es posible que el usuario no tenga permisos suficientes.
404	No encontrado	El recurso solicitado no existe.
500	Error interno del servidor	Se produjo un error interno del servidor al procesar la solicitud.

Ahora, pasaremos a explicar en qué casos usamos cada código de error y con qué finalidad.

- 200 OK: Este código generalmente se utiliza en los casos en que damos respuesta a solicitudes HTTP GET exitosas, para indicarle al cliente que la solicitud que realizó ha tenido éxito.
- 201 CREATED: Este código se utiliza cuando se realiza una petición HTTP POST o HTTP PUT, y se crea o modifica un nuevo recurso de forma exitosa, en ese sentido, consideramos apropiado devolver este código que indica que un nuevo recurso ha sido creado.

- 204 NO CONTENT: Cuando se realiza una petición HTTP DELETE, y la petición se ha completado con éxito pero su respuesta no tiene ningún contenido (debido a que el objeto se ha eliminado).
- 400 BAD REQUEST: Generalmente, este código de error lo utilizamos cuando se realiza una solicitud HTTP POST o HTTP PUT con alguno de los campos requeridos de forma inválida, en este sentido, nos parece apropiado indicar que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.
- 401 UNAUTHORIZED: Este código de error lo usamos para impedir que un usuario que no se encuentre logueado pueda hacer uso de las funcionalidades que requieren autenticación.
- 403 FORBIDDEN: Este código de error lo utilizamos cuando verificamos que un usuario esté logueado para que pueda realizar una determinada funcionalidad que requiere autenticación, y se tiene que el token recibido por header no es válido.
- 404 NOT FOUND: Este código de error lo utilizamos cuando se realiza una petición HTTP GET y no se encuentra el recurso solicitado, así como también en los casos en que se realiza un HTTP POST o HTTP PUT para crear o modificar un recurso que tiene asociado otro recurso, y no es posible encontrar ese recurso.
- 500 INTERNAL SERVER ERROR: Este código de error es utilizado cuando ocurre un problema interno, como es el caso de un problema en la base de datos.

## 5.3. Descripción de los resources de la API

### 5.3.1. BestOptionController

POST	/api/bestOption
Resources	Markets
Description	Recibe una lista de productos con su respectiva cantidad a adquirir, la distancia maxima a la que se desea buscar supermercados, la longitud y latitud del usuario, y el token de sesión, y retorna una lista ordenada por precio (ascendente) de cada supermercado que ofrece los productos deseados incluyendo: nombre y dirección del supermercado, cantidad de horas restantes para que el establecimiento cierre y el precio total para los productos seleccionados.
Body	<pre>{   "productId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",   "quantity": 2 }</pre>
Parameters	Recibe la longitud y latitud del usuario, y la distancia máxima a la cual quiere buscar supermercados (en kilómetros).
Responses	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y retorna una lista ordenada por precio (ascendente) de cada supermercado que ofrece los productos deseados incluyendo: nombre y dirección del supermercado, cantidad de horas restantes para que el establecimiento cierre y el precio total para los productos seleccionados.</p> <p><b>B) 401:</b> Ocurre cuando se intenta realizar la operación sin estar logueado en el sistema.</p> <p><b>C) 404:</b> Ocurre cuando no hay ninguna opción de supermercado disponible para los productos y/o distancia maxima seleccionada.</p> <p><b>D) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
Headers	Recibe un header adicional a los por defecto con el token de la sesión del usuario que se encuentra realizando la solicitud.

### 5.3.2. CategoryController

<b>GET</b>	/api/categories
<b>Resources</b>	Category
<b>Description</b>	Retorna una lista con las categorías de productos existentes en el sistema.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y retorna una lista con las categorías de productos en el sistema.</p> <p><b>B) 404:</b> Ocurre cuando no se encuentra ninguna categoría de producto disponible en el sistema.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningún header adicional a los por defecto.

### 5.3.3. ProductController

<b>GET</b>	/api/products
<b>Resources</b>	Product
<b>Description</b>	Obtiene todos los productos que se encuentran a la venta disponibles en los diferentes mercados, siempre y cuando la cantidad de ellos sea mayor a 0. En caso, de existir al menos un producto a la venta en un mercado, lo retorna.
<b>Parameters</b>	No recibe parámetros.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y devuelve la lista de productos disponible en los mercados.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, es decir, no existen productos a la venta.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningún header adicional a los por defecto.

<b>GET</b>	/api/products/productsByBarcode/
<b>Resources</b>	Product
<b>Description</b>	Realiza la búsqueda de un producto mediante el código de barra capturado por el escaner, el cual es pasado por parámetro. En caso de que exista lo retorna.
<b>Parameters</b>	Recibe el código de barras por el cual se quiere obtener el producto.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y se devuelve el producto buscado por código de barras.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, es decir, no existe el producto con el código de barras buscado.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningún header adicional a los por defecto.

<b>GET</b>	/api/products/productsByText/
<b>Resources</b>	Product
<b>Description</b>	Realiza la búsqueda de un producto mediante el texto buscado o categoría sobre un cierto criterio seleccionado. En caso de que exista al menos un producto, los retorna.
<b>Parameters</b>	Recibe el texto o categoría por el cual se quiere obtener los productos donde el nombre coincide con el texto buscado o la categoría seleccionada. Así como tambien el criterio por el cual el resultado se ordena.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y devuelve la lista de productos que coincidan con la búsqueda ordenados por el criterio seleccionado.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, es decir, no existen productos con el texto o categoría buscados.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningún header adicional a los por defecto.

<b>GET</b>	/api/products/discountProducts/
<b>Resources</b>	Product
<b>Description</b>	Realiza la búsqueda de todos los productos que tienen descuento, es decir, aquellos cuyo precio actual es menor al precio regular. En caso de existir, los retorna.
<b>Parameters</b>	No recibe parámetros.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y devuelve la lista de productos con descuento.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, es decir, no existen productos con descuento.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningún header adicional a los por defecto.

### 5.3.4. PurchaseController

<b>POST</b>	/api/purchases
<b>Resources</b>	Purchase
<b>Description</b>	Guarda y retorna una nueva compra asociada al usuario que tiene el token de sesión pasado en el header. Esto se realiza siempre y cuando, los campos sean correctos y el usuario se encuentre logueado en el sistema.
<b>Body</b>	{         "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",         "amount": 0,         "marketName": "string",         "marketAddress": "string",         "purchaseDate": "2021-06-09T20:06:54.596Z"     }
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y se guarda la compra asociada al usuario con el token de sesión pasado en el header, con la información contenida en el body. Una vez creada, la misma se retorna.</p> <p><b>B) 400:</b> Ocurre cuando el servidor no puede procesar la petición debido a algo que es percibido como un error del cliente, y muestra un mensaje indicado el error ocurrido.</p> <p><b>C) 401:</b> Ocurre cuando se intenta realizar la operación sin estar logueado en el sistema.</p> <p><b>D) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	Recibe un header adicional a los por defecto con el token de la sesión del usuario que se encuentra realizando la solicitud.

### 5.3.5. SessionController

<b>POST</b>	/api/sessions
<b>Resources</b>	Session
<b>Description</b>	Crea y retorna una nueva sesión de usuario con la información del usuario a loguear contenida en el body (obligatorio). Esto se realiza siempre y cuando los campos sean correctos.
<b>Body</b>	{           "username": "string",           "password": "string"         }
<b>Responses</b>	<p><b>A) 201:</b> Ocurre cuando la petición es exitosa y se crea la sesión de usuario con la información contenida en el body. Una vez creada, se devuelve el token de la sesión.</p> <p><b>B) 400:</b> Ocurre cuando el servidor no puede procesar la petición debido a algo que es percibido como un error del cliente, y muestra un mensaje indicado el error ocurrido.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningun header adicional a los por defecto

<b>DELETE</b>	/api/sessions/{token}
<b>Resources</b>	Session
<b>Description</b>	Elimina la sesión de usuario asociada al token pasado por parametro. Esto siempre y cuando sea un token valido.
<b>Parameters</b>	Recibe el token de la sesión a eliminar.
<b>Responses</b>	<p><b>A) 204:</b> Ocurre cuando la petición es exitosa y se ha eliminado correctamente la sesión. No muestra contenido.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, en este caso cuando no existe la sesión a eliminar.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningun header adicional a los por defecto

### 5.3.6. TokenController

<b>POST</b>	/api/tokens
<b>Resources</b>	Token
<b>Description</b>	Crea y retorna el token con la información contenida en el body (obligatorio). Esto se realiza siempre y cuando llegue el valor del token
<b>Body</b>	{           "tokenValue": "string",         }
<b>Responses</b>	<b>A) 201:</b> Ocurre cuando la petición es exitosa y se crea el token con la información contenida en el body. Una vez creado, el mismo se retorna. <b>B) 400:</b> Ocurre cuando el servidor no puede procesar la petición debido a algo que es percibido como un error del cliente, y muestra un mensaje indicado el error ocurrido. <b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.
<b>Headers</b>	No recibe ningun header adicional a los por defecto

### 5.3.7. UserController

<b>POST</b>	/api/users/
<b>Resources</b>	Users
<b>Description</b>	Crea y retorna un nuevo usuario con la información contenida en el body (obligatorio). Esto se realiza siempre y cuando, los campos sean correctos.
<b>Body</b>	{           "name": "string",           "username": "string",           "password": "string"         }
<b>Responses</b>	<b>A) 201:</b> Ocurre cuando la petición es exitosa y se crea el usuario con la información contenida en el body. Una vez creada, el mismo se retorna. <b>B) 400:</b> Ocurre cuando el servidor no puede procesar la petición debido a algo que es percibido como un error del cliente, y muestra un mensaje indicado el error ocurrido. <b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.
<b>Headers</b>	No recibe ningun header adicional a los por defecto

<b>GET</b>	/api/users/
<b>Resources</b>	Users
<b>Description</b>	Devuelve todos los usuarios existentes en el sistema.
<b>Parameters</b>	No recibe parámetro.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y se devuelven todos los usuarios.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, es decir, no hay usuarios registrados.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningun header adicional a los por defecto

<b>GET</b>	/api/users/{id}
<b>Resources</b>	Users
<b>Description</b>	Realiza la búsqueda de un usuario en el sistema a partir del id ingresado, y en caso de que exista lo retorna.
<b>Parameters</b>	Recibe el id del usuario a buscar.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y se devuelve el usuario que posee el id parámetro.</p> <p><b>B) 404:</b> Ocurre cuando no existe el recurso solicitado, es decir, no existe un usuario con el id buscado.</p> <p><b>C) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	No recibe ningún header adicional a los por defecto.

### 5.3.8. FavoritesController

<b>POST</b>	/api/favorites
<b>Resources</b>	Product
<b>Description</b>	Agrega un producto como favorito a la lista de productos favoritos del usuario autenticado.
<b>Parameters</b>	Recibe el ID del producto que se desea agregar como favorito al usuario logueado.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y se agrega el producto a la lista de productos favoritos del usuario.</p> <p><b>B) 401:</b> Ocurre cuando se intenta realizar la operación sin estar logueado en el sistema.</p> <p><b>C) 404:</b> Ocurre cuando no se encuentra el producto con el ID pasado por parámetro.</p> <p><b>D) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	Recibe un header adicional a los por defecto con el token de la sesión del usuario que se encuentra realizando la solicitud.

<b>GET</b>	/api/favorites
<b>Resources</b>	Product
<b>Description</b>	Retorna una lista con todos los productos marcados como favoritos del usuario fogueado
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y retorna una lista con todos los productos marcados como favoritos por el usuario logueado.</p> <p><b>B) 401:</b> Ocurre cuando se intenta realizar la operación sin estar logueado en el sistema.</p> <p><b>C) 404:</b> Ocurre cuando el usuario no tiene ningún producto marcado como favorito.</p> <p><b>D) 500:</b> Ocurre cuando sucede un problema interno en el servidor.</p>
<b>Headers</b>	Recibe un header adicional a los por defecto con el token de la sesión del usuario que se encuentra realizando la solicitud.

<b>DELETE</b>	/api/favorites
<b>Resources</b>	Product
<b>Description</b>	Elimina el producto con ID pasado por parámetro de la lista de productos favoritos del usuario logueado
<b>Parameters</b>	Recibe el ID del producto que el usuario logueado desea eliminar de su lista de productos favoritos.
<b>Responses</b>	<p><b>A) 200:</b> Ocurre cuando la petición es exitosa y se elimina el producto con ID pasado por parámetro de la lista de productos favoritos del usuario logueado.</p> <p><b>B) 401:</b> Ocurre cuando se intenta realizar la operación sin estar logueado en el sistema.</p> <p><b>C) 404:</b> Ocurre cuando el ID del producto pasado por parámetro no se corresponde con ningún producto del sistema.</p>
<b>Headers</b>	Recibe un header adicional a los por defecto con el token de la sesión del usuario que se encuentra realizando la solicitud.

# 6. Clean Code

Es muy importante que el código del proyecto cumpla con las condiciones mencionadas en el libro de Clean Code. Esto es debido a que el código será mantenido en el futuro en la gran mayoría de los proyectos y para poder mantener el código, este debe ser entendible y claro, y si se siguen las pautas de Clean Code esto se logra de forma fácil. Según el libro de Clean Code, hasta el mal código puede funcionar, pero que si no está limpio puede dar mucho trabajo a la hora de editarlo, mantenerlo y reorganizarlo. Todos los años se pierden muchas horas por culpa del código sucio. Robert C. Martin, alias Uncle Bob, escribió este libro para que mantener código no sea un trabajo difícil sino que sea tan fácil como agradable de hacer. En el libro, los primeros 10 capítulos nos cuentan sobre las principales pautas a seguir. En esta sección mostraremos ejemplos de nuestro código como forma de evidencia que el desarrollo fue realizado siguiendo clean code.

## 6.1. Nombres nemotécnicos

Clean Code nos dice que la elección de los nombres es claramente importante para el trabajo y el entendimiento del código. Deben revelar la intención de las cosas. La nomenclatura se definió para que al leer el nombre de métodos, variables, entre otros, se pueda comprender fácilmente cuál es su función a cumplir y que no haya lugar a confusiones o ambigüedades, evitamos dar nombres que den ideas erradas y contribuyan a la desinformación, y los nombres utilizados son sencillos y pronunciables.

```
public string Name { get; set; }

public string Address { get; set; }

public float Longitude { get; set; }
|
public float Latitude { get; set; }
```

Figura 6.1: Ejemplos de nombres nemotécnicos utilizados en la solución.

## 6.2. Nombre de clases y métodos

Para los nombres de las clases, no utilizamos verbos, ni palabras no significativas o genéricas.

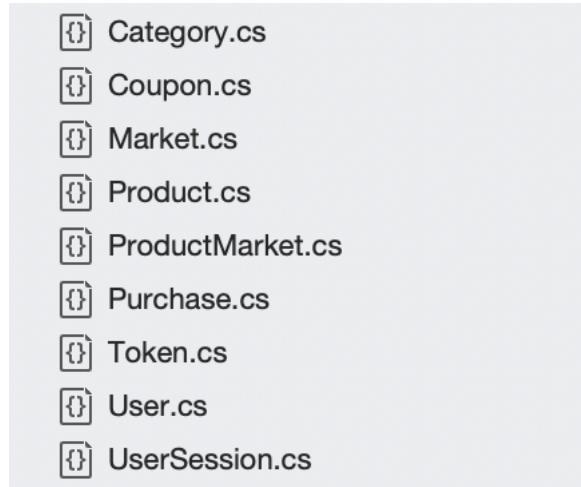


Figura 6.2: Ejemplos de nombres adecuados en clases de la solución.

En los nombres de los métodos siempre incluimos un verbo, e intentamos ser consistentes en el sentido de que por ejemplo, todos los métodos que se encargan de obtener elementos poseen la palabra Get.

```
public List<Coupon> GetAllFromUser(Guid token)
{
    var userId = userSessionRepository.GetTokenUserId(token);
    var user = userRepository.GetUserByIdWithCoupons(userId);
    if (user == null)
    {
        throw new DomainBusinessLogicException(MessageExceptionBusinessLogic.ErrorObtainedAllCouponsOfUser);
    }
    return user.Coupons;
}
```

Figura 6.3: Ejemplo de utilización de un método que incluye la palabra 'get' mostrando que trae ciertos elementos.

```
private void Generate(Guid token, Market market, DateTime deadline, int value)
{
    var userId = userSessionRepository.GetTokenUserId(token);
    var user = userRepository.Get(userId);
    Coupon newCoupon = new Coupon() { Market = market, Deadline = deadline, Value = value };
    user.Coupons.Add(newCoupon);
    userRepository.Update(user);
}
```

Figura 6.4: Ejemplo de utilización de verbo en un método de la solución.

## 6.3. Funciones

Las funciones que desarrollamos, están enfocadas a factores que hacen que la función sea fácil de leer y de entender, y para que a futuro se entienda bien cómo están hechas, y cómo modificarlas.

En general las funciones desarrolladas son pequeñas, hacen una sola cosa (no están divididas en secciones), utilizan un solo nivel de abstracción, es decir, por ejemplo en una función perteneciente a una clase de la capa de las reglas de negocios no introducimos operaciones relativas al acceso a la base de datos. Esto lo logramos como consecuencia de la clara independencia de capas la cual se encuentra justificada con más detalle en la sección de Descripción de Diseño.

Siguiendo los lineamientos de Clean Code, podemos afirmar que las funciones poseen el nivel de indentación correcto, y en la gran mayoría de los casos el mismo no es mayor a uno o dos.

También, podemos destacar que debido a la forma en que están implementadas las funciones, es posible leer el código *from top to bottom*, lo que significa que debajo de una función, se encuentran las funciones que son utilizadas por ella.

Un ejemplo de función mostrando algunos de los aspectos mencionados previamente:

```
public User Create(User newData)
{
    try
    {
        newData.Id = Guid.NewGuid();
        newData.Validate();
        VerifyIfUserExist(newData);
        userRepository.Add(newData);
        return newData;
    }
    catch (UserException e)
    {
        throw new DomainBusinessLogicException(e.Message);
    }
    catch (DomainBusinessLogicException e)
    {
        throw new DomainBusinessLogicException(e.Message);
    }
}
private void VerifyIfUserExist(User user)
{
    User userObtained = userRepository.GetUserByUsername(user.Username);
    if (userObtained != null)
    {
        throw new DomainBusinessLogicException(MessageExceptionBusinessLogic.ErrorUserAlredyExist);
    }
}
```

Figura 6.5: Ejemplo de una función en donde es posible leer el código *from top to bottom*.

Se puede observar como la función *VerifyIfUserExist* es utilizada en la función *Create*, y se encuentra ubicada debajo de la misma, permitiendo una lectura del código *from top to bottom*.

## **6.4. Comentarios**

Siguiendo los principios de Clean Code, que nos indica que el código idealmente no debería estar comentado, nuestro código no posee ningún comentario.

No posee comentarios porque consideramos que el código es autoexplicativo. Si tuviéramos que escribir un comentario, significa que el código no es suficientemente autoexplicativo, y en ese sentido deberíamos poder mejorarlo en lugar de tapar la falencia con comentarios.

Los comentarios que son tolerables para Clean Code son aquellos que refieren a aspectos legales, aclaraciones que el código no puede ser modificado por terceros, entre otros. Estos tipos de comentarios no consideramos apropiados incorporarlos a nuestra solución porque no aportan valor agregado alguno en nuestro caso.

## **6.5. Formateo**

Al comenzar el desarrollo, nos inclinamos por escoger un grupo de reglas simples que se encuentren en todo el formato del código, para mantener una determinada consistencia.

### **6.5.1. Horizontal**

En este sentido, se refiere a cómo está organizado el código, es decir, el largo de los archivos, el ancho de los archivos, la buena indentación entre otras cosas. En la próxima imagen podemos ver el formato de un archivo, el cual no es excesivamente grande horizontalmente:

```

        listOfCategories.Add(butcherShopCategory);

    var categoryMock = new Mock<IRepository<Category>>(MockBehavior.Strict);
    categoryMock.Setup(m => m.GetAll()).Throws(new ServerException());

    CategoryManagement categoryLogic = new CategoryManagement(categoryMock.Object);

    categoryLogic.GetAllCategories();
}

[TestMethod]
[ExpectedException(typeof(ClientBusinessLogicException))]
public void GetClientExceptionObtainedAllCategoriesTest()
{
    List<Category> listOfCategories = new List<Category>();
    Category bakeryCategory = new Category
    {
        Id = Guid.NewGuid(),
        Name = "Panadería"
    };
    listOfCategories.Add(bakeryCategory);
    Category butcherShopCategory = new Category
    {
        Id = Guid.NewGuid(),
        Name = "Carnicería"
    };
    listOfCategories.Add(butcherShopCategory);

    var categoryMock = new Mock<IRepository<Category>>(MockBehavior.Strict);
    categoryMock.Setup(m => m.GetAll()).Throws(new ClientException());

    CategoryManagement categoryLogic = new CategoryManagement(categoryMock.Object);

    categoryLogic.GetAllCategories();
}

```

Figura 6.6: Ejemplo de formateo horizontal

### 6.5.2. Vertical

El formateo vertical hace referencia a diferentes conceptos. Uno de ellos es la llamada metáfora del periodico, que indica que el nombre de la clase debe ser simple pero explicativo, y se incrementará en los detalles mientras se recorre el código hacia abajo. Como se mostró previamente, los nombres son simples, y para obtener más detalles de la clase, se va recorriendo el código hacia abajo y se los va obteniendo.

Otro aspecto que se destaca en este punto, es que Clean Code recomienda que debemos declarar las variables lo más cerca posible de su uso.

Un ejemplo de aplicación de este concepto es el siguiente:

```

public List<Coupon> GetAllFromUser(Guid token)
{
    var userId = userSessionRepository.GetTokenUserId(token);
    var user = userRepository.GetUserByIdWithCoupons(userId);
    if (user == null)
    {
        throw new DomainBusinessLogicException(MessageExceptionBusinessLogic.ErrorObtainedAllCouponsOfUser);
    }
    return user.Coupons;
}

```

Figura 6.7: Ejemplo de declaración de variables lo mas cercano posible a su uso.

Se puede apreciar como se obtiene el usuario inmediatamente antes de que se

consulte si el mismo es *null*.

Otra lineamiento que se describe en este sentido, es que las instancias de las variables, deben estar al principio de la clase. Esto no debe ser impedimento a la distancia de su uso, ya que en una clase bien ideada, se usan las variables en la mayoría (si no en todos), los métodos de la misma.

En la siguiente imagen tenemos una de las clases de nuestra aplicación, en donde se puede observar claramente que las instancias de las variables están ubicadas al principio de la misma.

```
public class Category
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public string ImageUrl { get; set; }

    public Category()
    { }

    public override bool Equals(object obj)
    {
        return obj is Category category &&
               Name.Equals(category.Name);
    }
}
```

Figura 6.8: Ejemplo en donde se muestra que las instancias de las variables se encuentran al principio de la clase.

## 6.6. Pruebas unitarias

Robert Martin habla de las tres leyes de las pruebas unitarias:

- No escribirás código de producción hasta haber escrito una prueba fallida.
- No escribirás más sobre una prueba unitaria que lo suficiente para que falle.
- No escribirás más código de producción que el necesario para que pase la prueba.

No podemos ofrecer una evidencia de completa del cumplimiento de estas leyes, pero por medio de los commits en el repositorio en GitHub, se puede visualizar correctamente debido a que no hay commits que contengan lógica que no esté acompañada por sus respectivos tests.

Clean Code también nos indica que las pruebas al tener que evolucionar al mismo ritmo que el código, deben ser igualmente mantenibles y respetar las mismas reglas de código limpio.

Está permitido hacer más de un assert en una prueba, pero sí se debe cumplir que el número de asserts sea mínimo. Lo que sí se debe cumplir siempre es que solo se prueba una cosa en cada test.

Nuestros métodos de tests en su totalidad incluyen un único assert, afirmando el concepto de que los mismos prueban una única cosa.

También, es importante notar que siguiendo Clean Code, intentamos evitar métodos de test muy largos con todos los detalles de implementación, intentando mostrar claramente la estructura Arrange-Act-Assert de las pruebas escondiendo los detalles en métodos.

```
[TestMethod]
0 | 0 referencias | Agustín, Hace 21 días | 2 autores, 2 cambios
public void TestGetMinimumAndMaximumPriceOk()
{
    Tuple<double, double> minAndMaxPrice = new Tuple<double, double>(30.0, 50.0);
    var productMarketMock = new Mock<IProductMarketRepository>(MockBehavior.Strict);
    productMarketMock.Setup(m => m.GetMinimumAndMaximumPrice(It.IsAny<Guid>())).Returns(minAndMaxPrice);
    IProductMarketManagement productMarketLogic = new ProductMarketManagement(productMarketMock.Object);
    Tuple<double, double> result = productMarketLogic.GetMinimumAndMaximumPrice(Guid.NewGuid());
    productMarketMock.VerifyAll();
    Assert.AreEqual(minAndMaxPrice, result);
}
```

Figura 6.9: Ejemplo de test que cumple con lo mencionado previamente.

# 7. Instructivo de instalación

## 7.1. Deployment Backend

1. Compilar la aplicación en Release.
2. Crear una publicación de nuestro sistema. Nos posicionamos sobre el paquete de WebApi y damos en publicar, dejamos la ruta por defecto.
3. Si vamos a la ruta “[WebApiProject]/bin/Release/netcoreapp5.0/” encontraremos una carpeta llamada “publish” y dentro estarán los archivos de nuestra aplicación así como también las bibliotecas de clases de otras dependencias como EFCore.
4. Copiamos todo estos archivos, y luego vamos al directorio “C:/inetpub/wwwroot” creamos una carpeta en dónde vamos a poner el backend, y dentro de la misma, pegamos todos los archivos que teníamos copiados.
5. Por defecto no podremos ejecutar peticiones del tipo PUT ni DELETE. Para ello debemos quitar el módulo de WebDAV de nuestra aplicación. Dentro de la carpeta “publish” generada, se encuentra un archivo llamado “web.config”, el contenido es similar a este:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <location path="/" inheritInChildApplications="false">
    <system.webServer>
      <handlers>
        <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModuleV2" resourceType="Unspecified" />
      </handlers>
      <aspNetCore processPath="dotnet" arguments=".\\WebApi.dll" stdoutLogEnabled="false" stdoutLogFile=".\\logs\\stdout" hostingModel="inprocess" />
    </system.webServer>
  </location>
</configuration>
<!--ProjectGuid: 54729B1D-96DF-4139-8534-4E41563C258A-->
```

Figura 7.1: Archivo web.config

Dónde en este código debemos agregar la siguiente configuración:

```
<modules runAllManagedModulesForAllRequests=false>
<remove name="WebDAVModule" />
</modules>
```

Quedando un archivo como el siguiente:



Figura 7.2: Archivo web.config modificado.

6. Ir a IIS y crear un nuevo sitio:

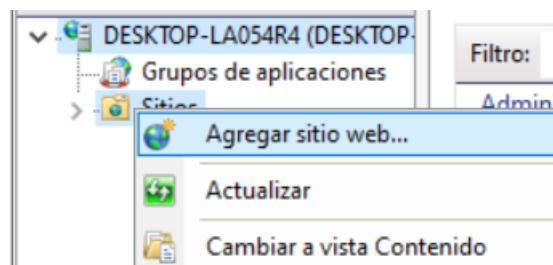


Figura 7.3: Creación de un nuevo sitio.

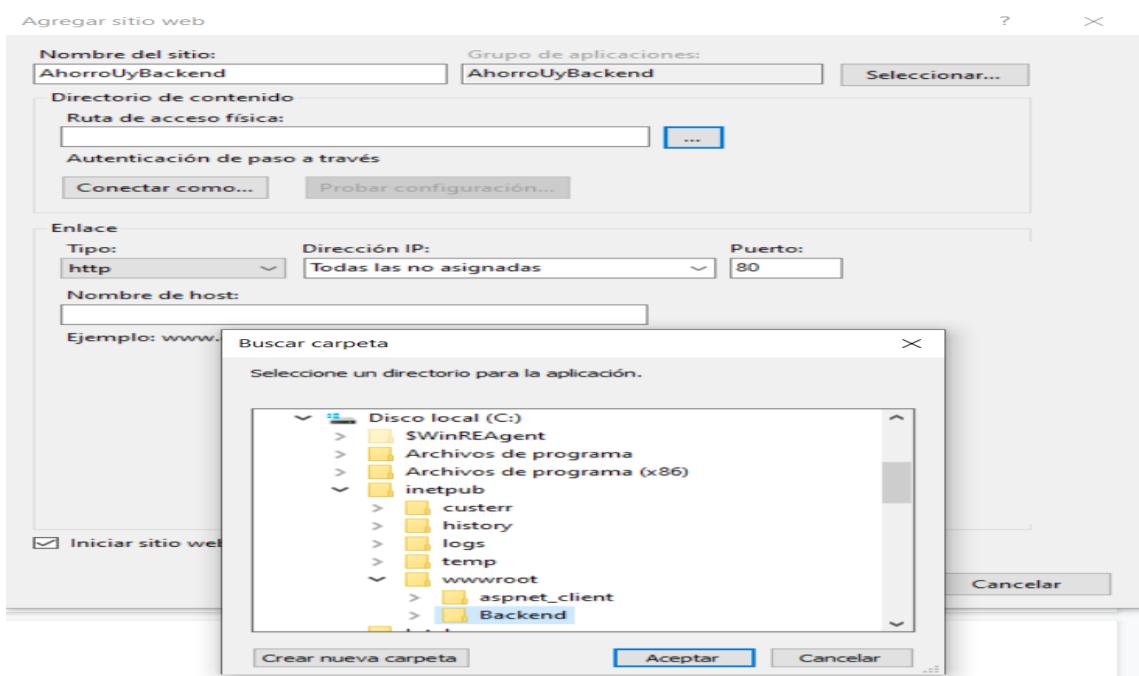


Figura 7.4: Configuración en la creación de un nuevo sitio.

- El nombre del sitio es a elección, en nuestro caso lo llamamos igual que la carpeta donde creamos el BackEnd, pero esto no es un requisito necesario.
- La ruta de acceso física es la ruta a nuestra carpeta donde copiamos los archivos anteriores que contiene la aplicación, esto en el directorio “C:/inetpub/wwwroot/Backend”
- El puerto es también a elección. Por defecto IIS cuenta con un sitio por defecto “Default Website” que ya corre en el puerto 80. Sólo podemos tener corriendo un único sitio por puerto. Pueden poner otro puerto o dejar el 80 y luego detener el sitio por Default. En nuestro caso elegimos el puerto 6969.
- Cuando creamos un nuevo sitio, IIS nos creará un nuevo POOL o grupo de aplicaciones para ese nuevo sitio, luego de esto nos dirigiremos a grupo de aplicaciones, y haremos doble click sobre el pool recién creado qué tendrá el mismo nombre que le pusimos al sitio.

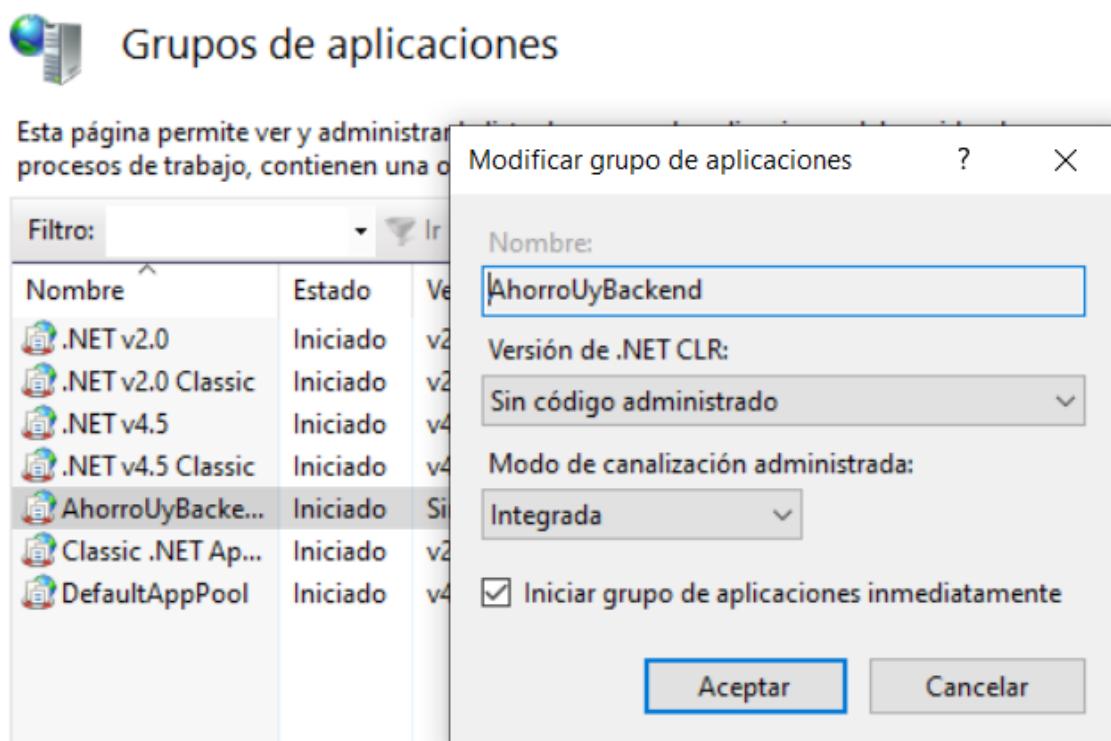


Figura 7.5: Archivo web.config

Aquí lo que haremos serán los siguientes pasos:

- a) Quitar el CLR por defecto, y poner sin código administrado. Esto ya que .NET Core no usa el CLR por defecto configurado en IIS.
- b) Otorgar permisos al usuario del sitio de IIS sobre la base. El usuario que interactúa contra el motor de SQL Server es el usuario

creado para el sitio de IIS que creamos anteriormente (IIS APPPOOL/XXXX). Debemos entonces configurar un nuevo inicio de sesión en el motor de SQL Server y otorgarle permisos para que pueda leer y escribir de a la base. Simplemente crearemos un usuario sysadmin.

- Ir a la carpeta de Security y crear un nuevo inicio de sesión

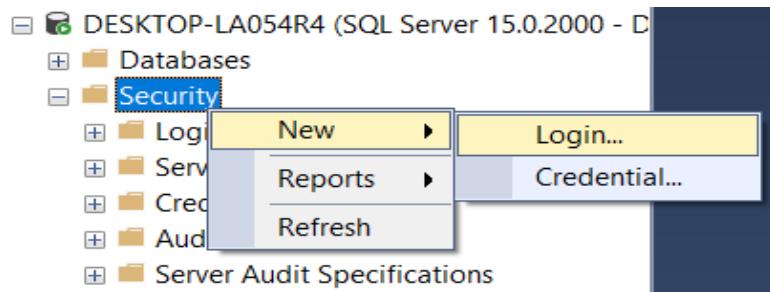


Figura 7.6: Creación de un nuevo inicio de sesión.

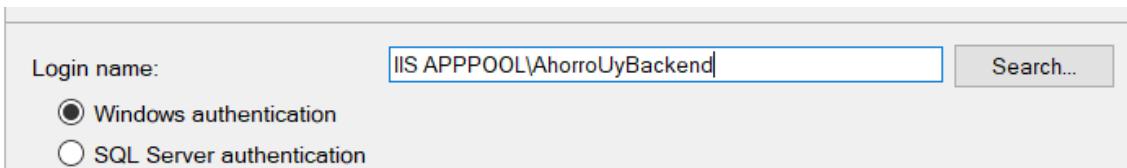


Figura 7.7: Nombre del login a introducir.

- A este nuevo usuario debemos asignarle el ROL deseado, en nuestro caso sysadmin. Una vez hecho esto, darle OK.

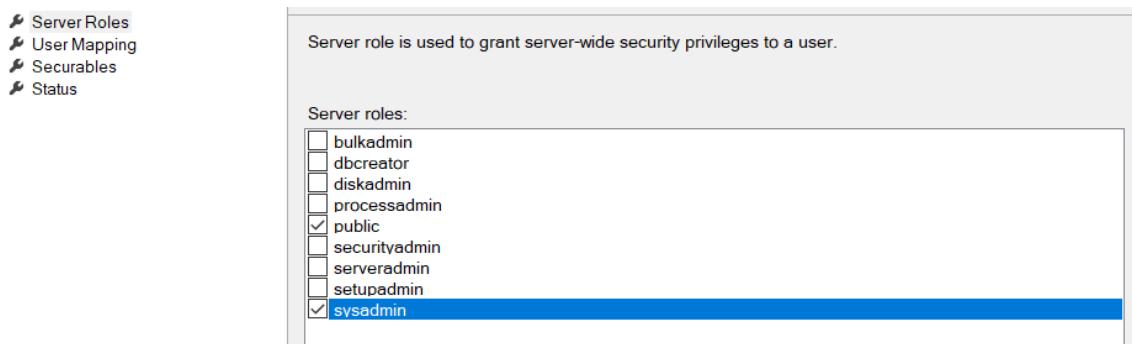


Figura 7.8: Asignación de rol sysadmin.

Luego de esto visualizamos si carga correctamente el backend, introduciendo en el navegador la siguiente URL “<http://localhost:6969>”, y sí carga correctamente significa que ha quedado funcionando de manera correcta.

## 7.2. Aplicación Android

Se debe tener en cuenta que en caso de tener que cambiar la URL base, se debe dirigir en el archivo Constants dentro del paquete API, y cambiar la variable BASEURL que allí se encuentra. Tal como se muestra en la siguientes imagenes.

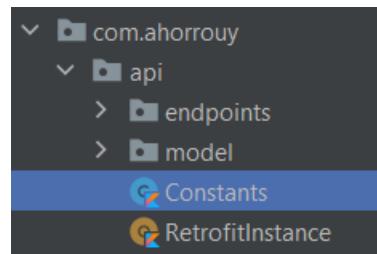


Figura 7.9: Carpeta donde se encuentra la clase Constants.

```
const val BASE_URL = "http://10.0.2.2:5000/api/"
```

Figura 7.10: Constante que se debe modificar.

# Bibliografía

- [1] Gestión de riesgos en scrum. [Online]. Available: <https://wolfproject.es/scrum-riesgos-gestion/>
- [2] Tabla comparativa de precios de canasta básica de alimentos al 10 de marzo - ministerio de economía y finanzas. [Online]. Available: <https://www.gub.uy/ministerio-economia-finanzas/comunicacion/noticias/tabla-comparativa-precios-canasta-basica-alimentos-10-marzo>
- [3] Sonarqube documentation. [Online]. Available: <https://docs.sonarqube.org/>
- [4] Designite documentation. [Online]. Available: <https://www.designite-tools.com/>
- [5] J. Sutherland and K. Schwaber, “The scrum guide,” *The definitive guide to scrum: The rules of the game. Scrum. org*, vol. 268, 2013.
- [6] A. Guide, “Project management body of knowledge (pmbok® guide),” in *Project Management Institute*, 2001.
- [7] M. Trigás Gallego, “Metodología scrum,” 2012.
- [8] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002, vol. 1.
- [9] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.