Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la computación

Diseño Detallado de Software - IIC 2113 - 2022-1

Entrega 5: Diseño Detallado de Software

Grupo 22

Integrantes:

- Vicente Espinosa
- Ignacio Zúñiga
- Alejandro Rico

Profesor:

- Antonio Ossa

Ayudante:

- Diego Cartagena

Resumen

El proyecto que elegimos fue el de *ecommerce que consistía en* desarrollar un *marketplace*, en nuestro caso realizamos una tienda de artículos electrónicos. La aplicación debía contar con un flujo básico de tienda online, donde existe un landing page con el listado de productos, que se puede filtrar por ciertos atributos, agregar items al carrito de compras, y dentro del carrito poder modificar las cantidades, y ver precios a pagar, con costos de envío y propinas. Otra funcionalidad fue la de incorporar cupones de descuento. Estos podrían aplicar ciertos porcentajes de descuento sobre el precio total del carrito o sobre artículos de categorías específicas. También la aplicación contaba con una integración a una API que convierte todos los precios a dólares. Por último, contamos con una sección donde los administradores de la tienda pueden gestionar las categorías de los artículos publicados.

La aplicación la desarrollamos con las tecnologías de Ruby en Rails para el Backend y ReactJs para el Frontend. Para la aplicación en Rails, optamos por utilizar una API REST, esto ayudó a prevenir la generación automática de archivos extras y el formato JSON de respuesta en los endpoints. Para el Frontend elegimos React porque es una tecnología conocida por los 3. La API utilizada para el cambio de divisas fue la de API Chile, donde se obtenía el último precio oficial del dólar en CLP, y se hacía el cambio en la página.

Los principales contenidos del curso que utilizamos durante este proyecto fueron: Diagramas de Diseño para preparar la implementación a distintos niveles, code reviews para evitar Code Smells, Refactoring a la hora de juntar nuestros proyectos individuales y Testing para lo solicitado en la entrega pasada.

Diagramas 4+1

Vista Lógica

Para la vista lógica como se ve en la figura 1, creamos un diagrama de clases que representan los modelos y entidades de la aplicación. Agregamos durante el proyecto el modelo UsedCoupons, esto debido a que era necesario separar la lógica de aplicar cupones y ver cupones disponibles con la de ver el historial de los cupones usados por alguien.

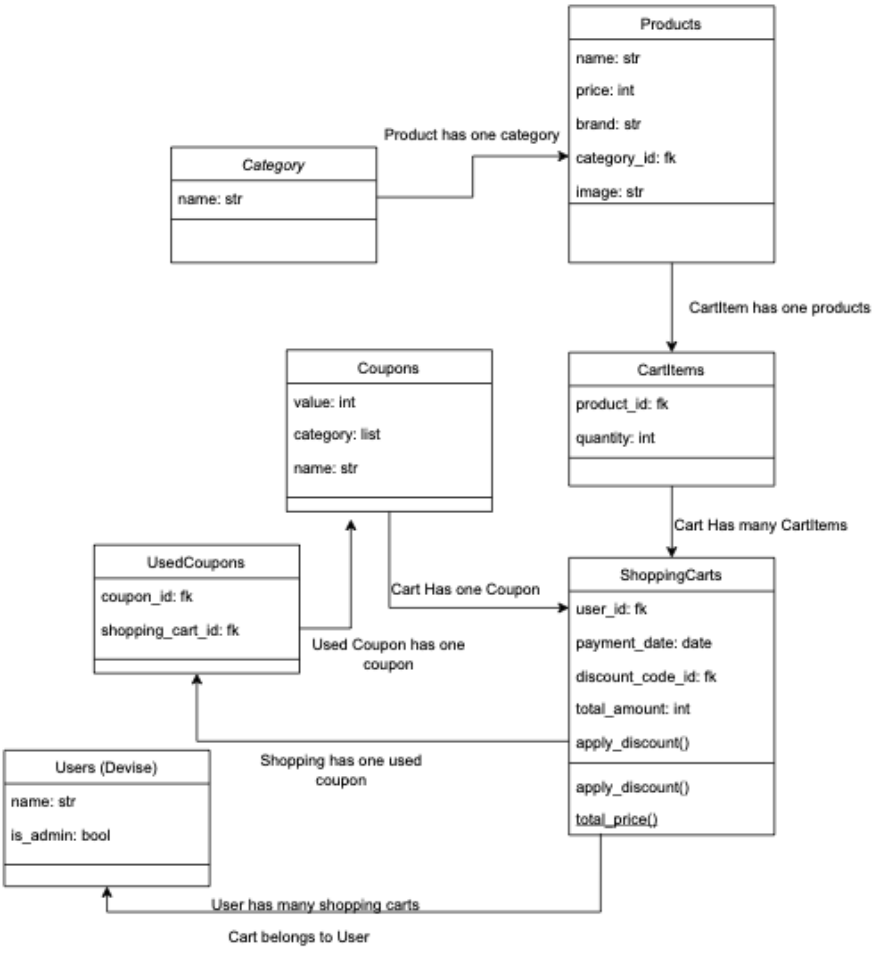

Figura 1: Diagrama de Clases

Vista de Implementación

Para la vista de implementación como se ve en la figura 2, usamos un diagrama de componentes, ya que este muestra un poco más alto nivel (en comparación al diagrama de clases) de cómo funciona la aplicación y cómo desarrollamos la arquitectura en grandes rasgos. Para este caso tenemos el backend separado del frontend y que la base de datos se conecta solamente al backend.

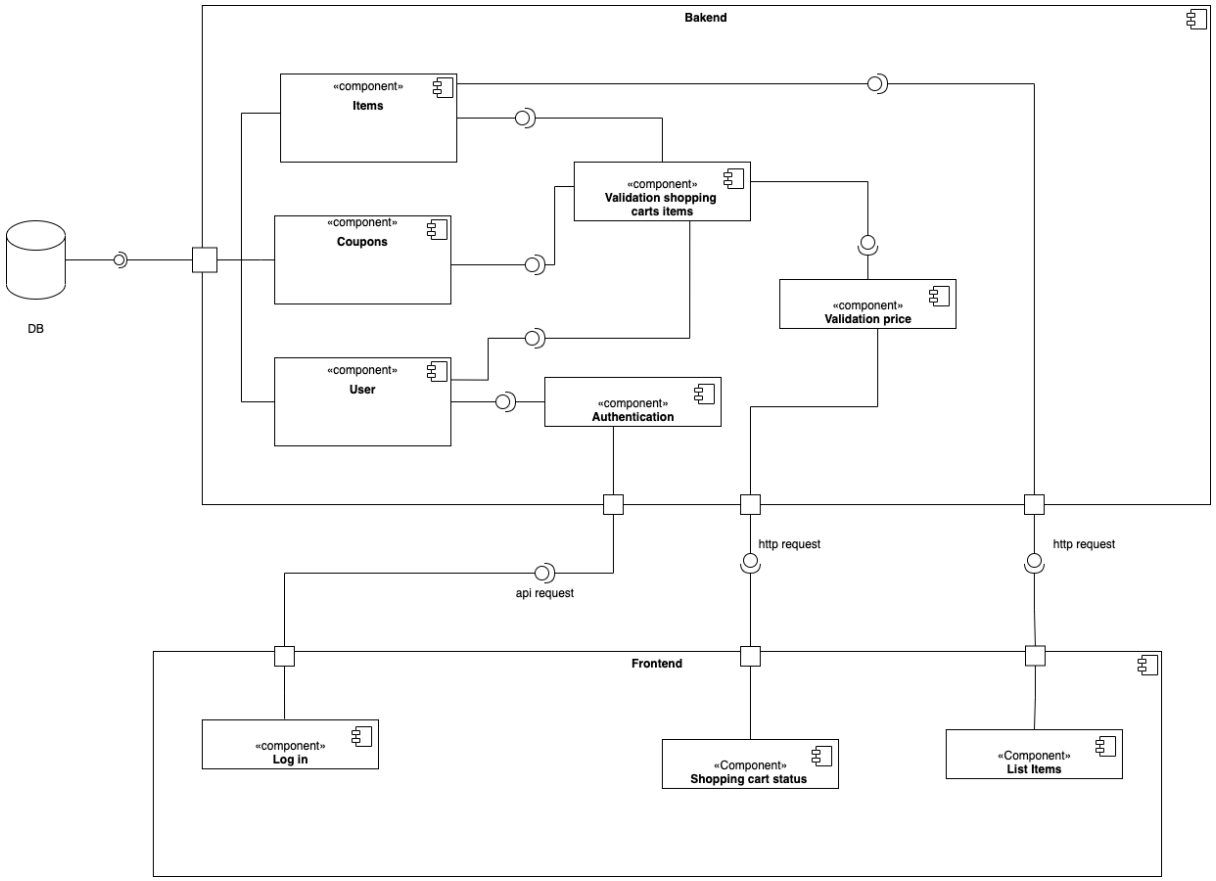

Figura 2: Diagrama de componentes

Vista de procesos

Para la vista de procesos como se ve en la figura 3, usamos un diagrama de despliegue, ya que muestra en paquetes cómo es la arquitectura de la aplicación. Acá separamos el servidor donde está el backend y la base de datos, con el cliente que contiene el frontend en react.

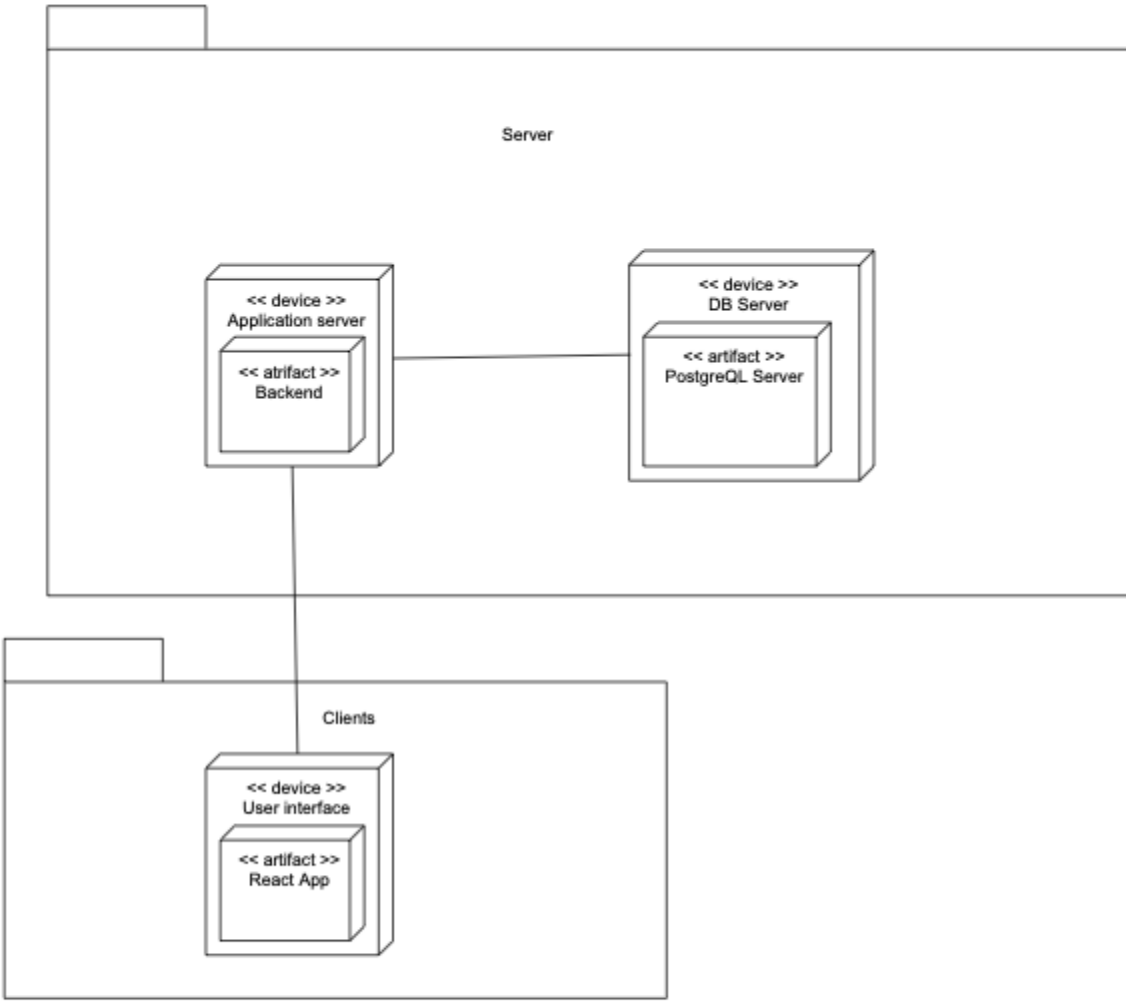

Figura 3: Diagrama de despliegue.

Vista Física

En este diagrama se pueden ver múltiples flujos que puede seguir un usuario en la página, terminando al momento de finalizar la compra. Se pueden ver los encargados de cada tarea.

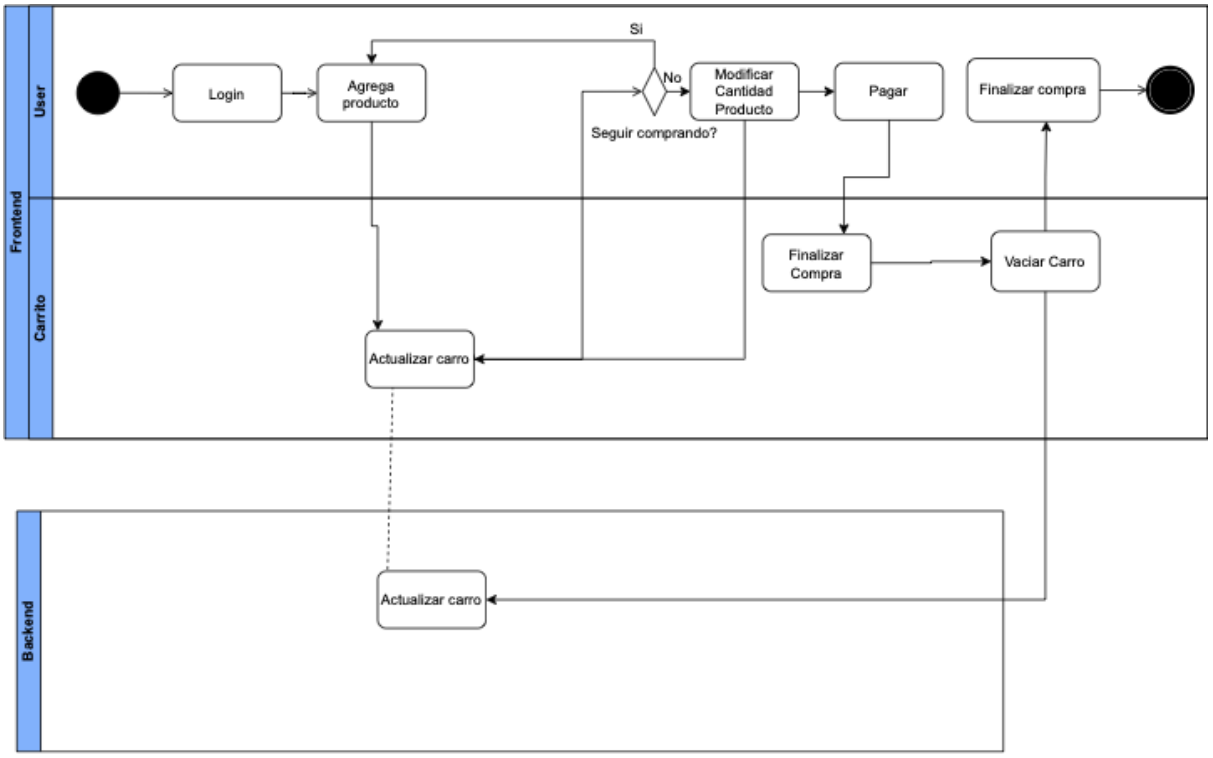

Figura 4: Diagrama de Actividad

Vista de escenarios

Para la vista de escenarios creamos un flujo básico que debe seguir un usuario para lograr una compra exitosa. Aquí se muestra tanto la interacción entre el usuario y el frontend como la del frontend con el backend.

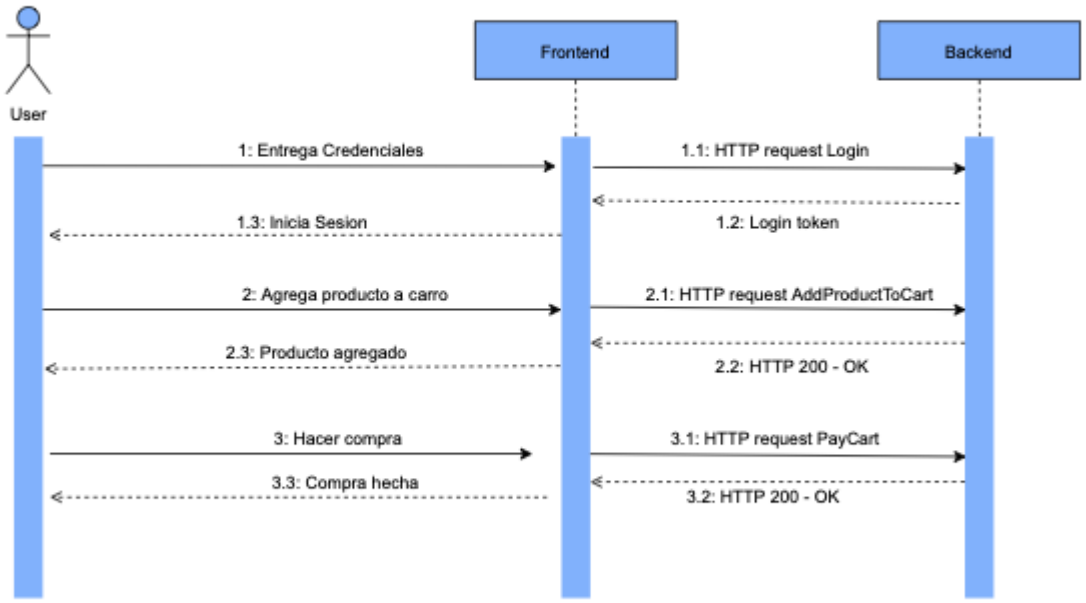

Figura 5: Diagrama de secuencia

MOCKUPS:

Mockup Lista de Productos

La figura 6 es la vista principal donde los usuarios pueden acceder a todos los productos disponibles. En la implementación e iteración, luego de realizar el mockup, decidimos incorporar un filtro de productos con una barra buscadora, en lugar de seleccionar las categorías como sale en la imagen y el botón de cambiar a dólar los precios. Además se incorporó el botón de agregar producto al carrito en la tarjeta de cada producto y las secciones de ‘cupones’ y ‘mis compras’ en la barra de navegación.

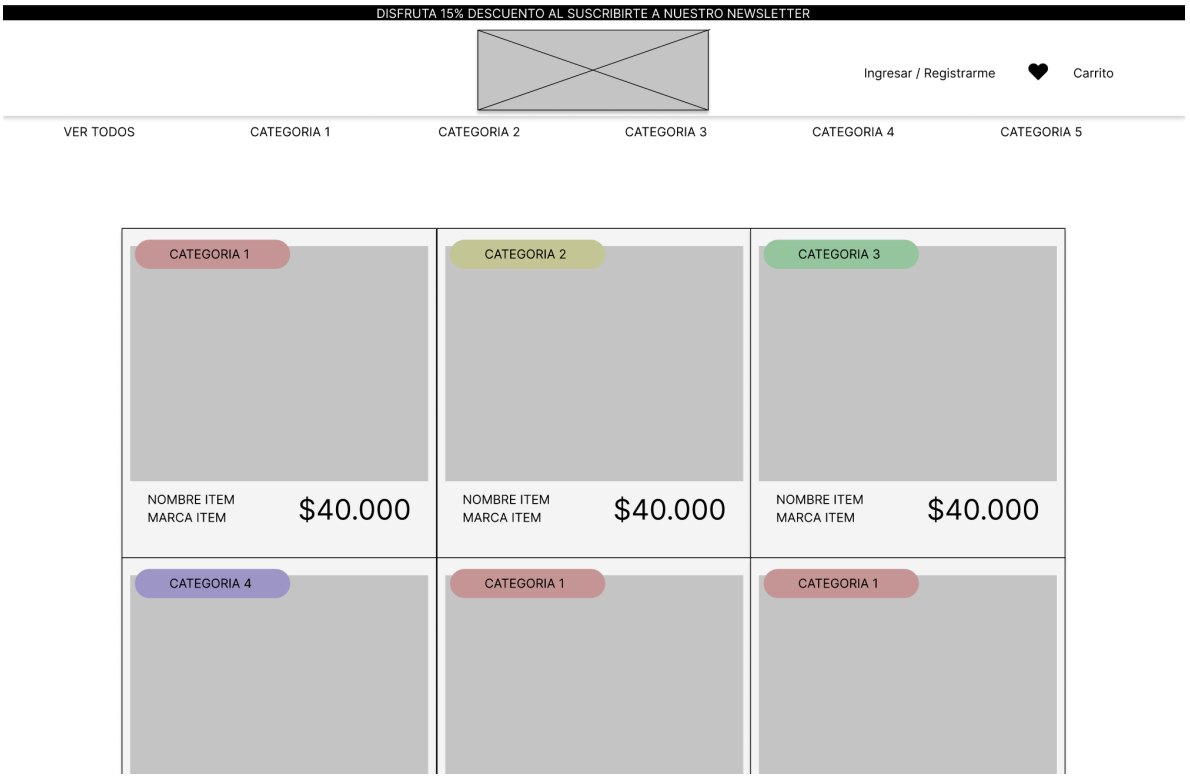

Figura 6: Lista de productos

Mockup Vista de Administración

La figura 7 es la vista de administrador, se accede escribiendo {url}/admin, aquí los usuarios administradores de la aplicación pueden visualizar los productos, crear, eliminar y modificar categorías. El diseño se mantuvo muy similar durante todo el trayecto, sin embargo, un detalle que cambió fue la barra buscadora, que al igual que en la vista anterior, no se filtra por categoría, sino por match de palabras en categorías, nombre del producto o marca.

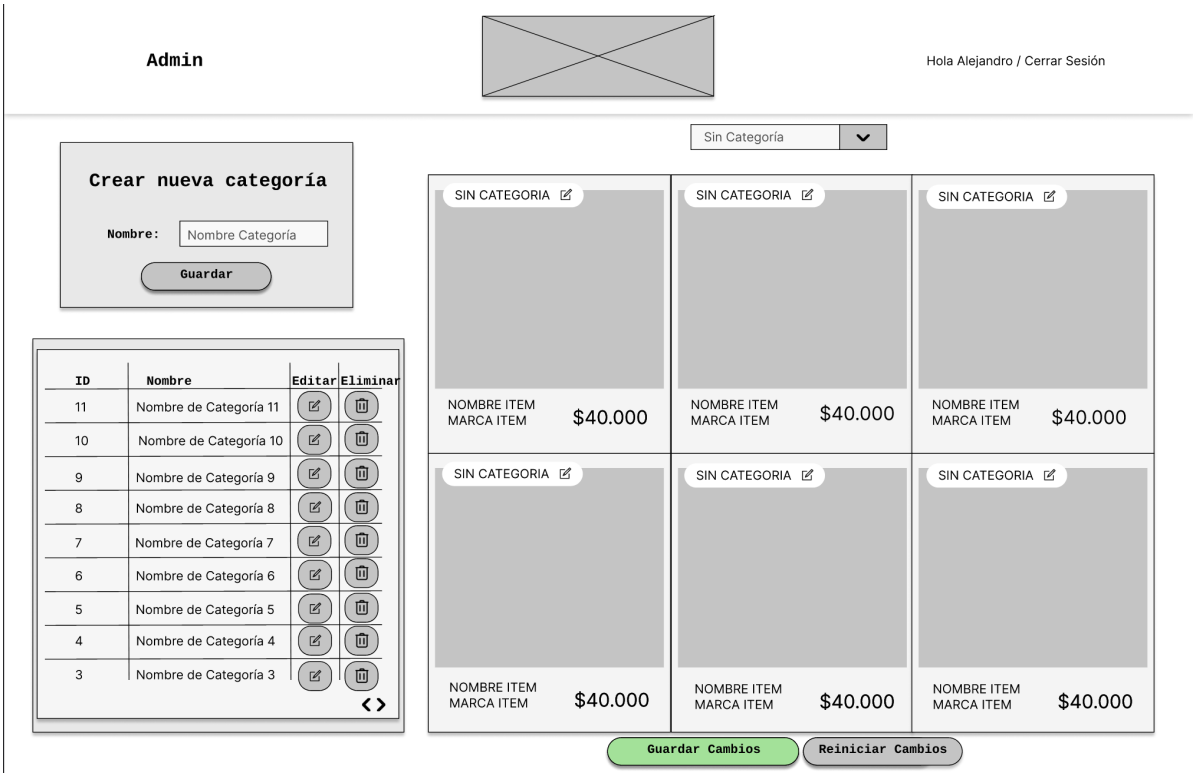

Figura 7: Vista de administrador

Mockup Cupones Usados

La figura 8 es una vista que muestra muestra los distintos cupones usados por el usuario, junto con las categorías con las que podría haber usado este mismo cupón y además muestra cuánto fue lo que ahorró al usarlo. Finalmente se decidió remover la descripción de los cupones debido a que no aportaba nada al proyecto.

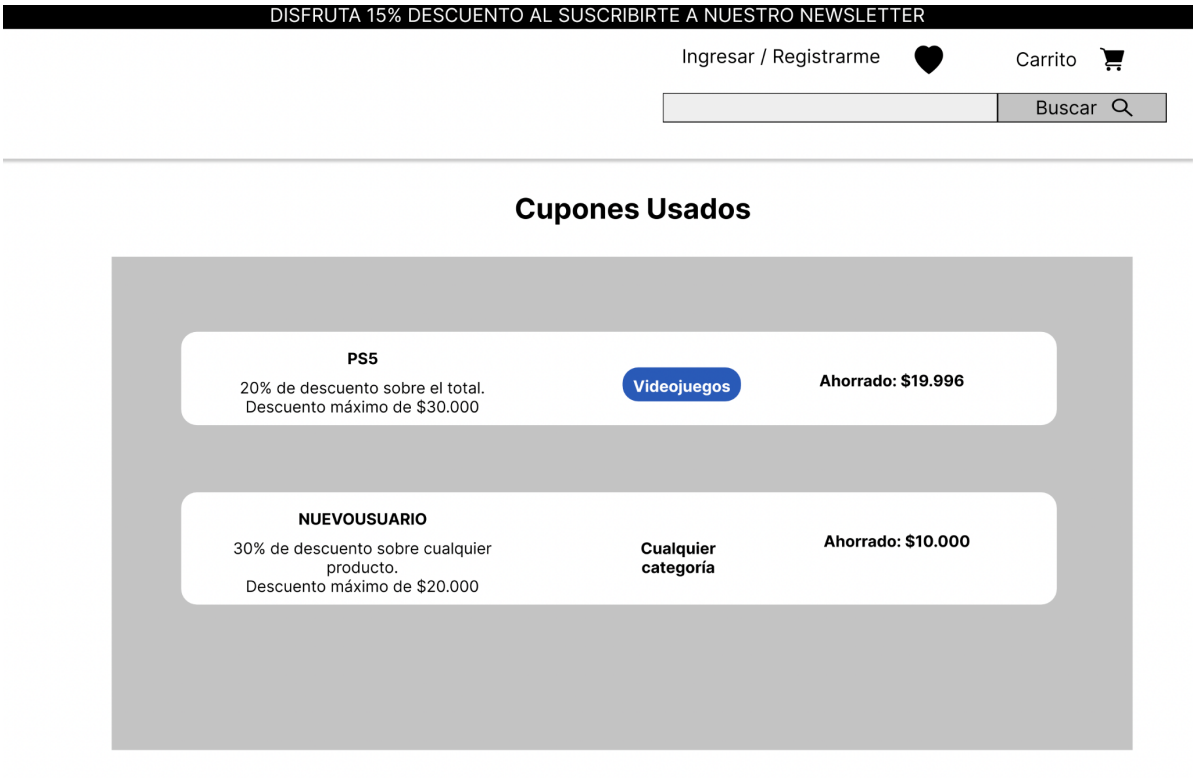

Figura 8: Vista de cupones usados

Mockup Vista Historial De Compras

La figura 9 es una vista que muestra el historial de compras pasadas del usuario, se puede ver el tipo de producto que se compró, junto con la cantidad que se compró de este mismo y el precio unitario. También se puede ver la fecha la compra, el total, y cuando aplica, el subtotal y cuanto se descontó con el cupón. Las funcionalidades de los botones “Ver compra” y “Volver a comprar” no se implementaron ya que se escapaba de lo solicitado por el proyecto.

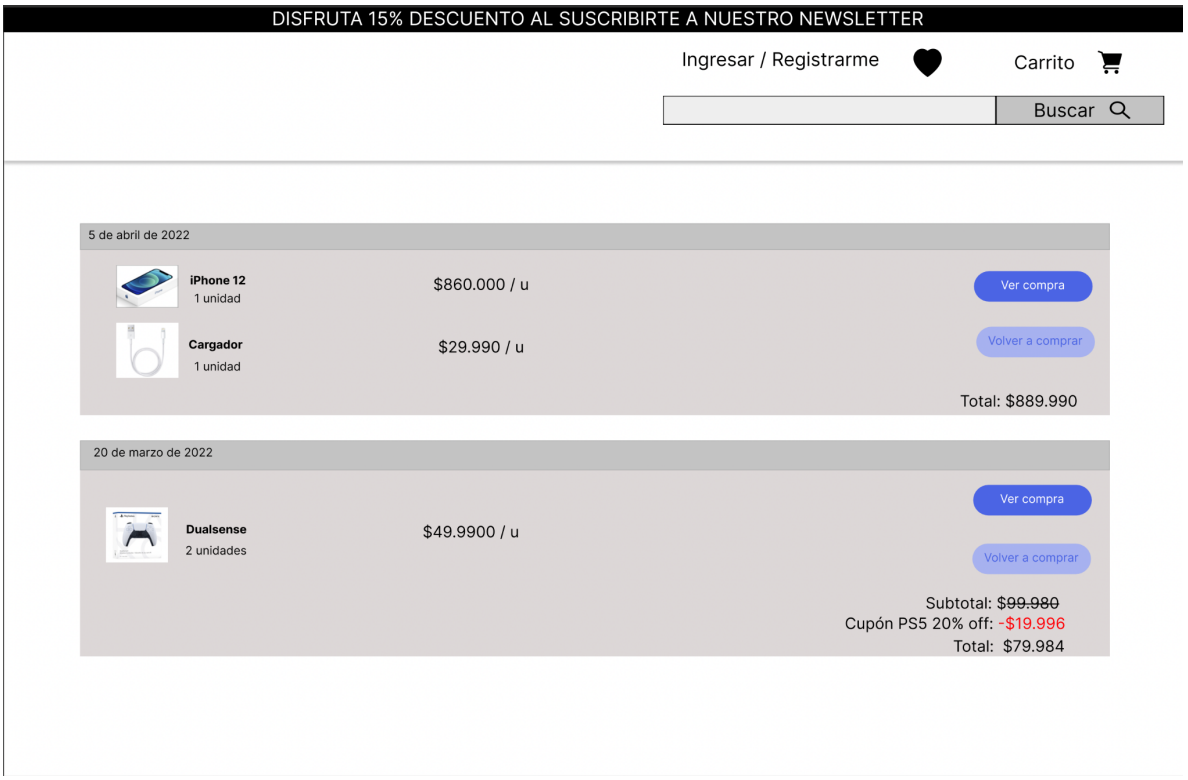

Figura 9: Vista de historial de compras

Mockup Vista Carro de Compras

La figura 10 muestra la vista del carro de compras. Acá se puede ver la lista de compras que está dentro del carro en la tabla de la izquierda, un resumen del precio final a la derecha y la barra de navegación en la parte superior de la imagen

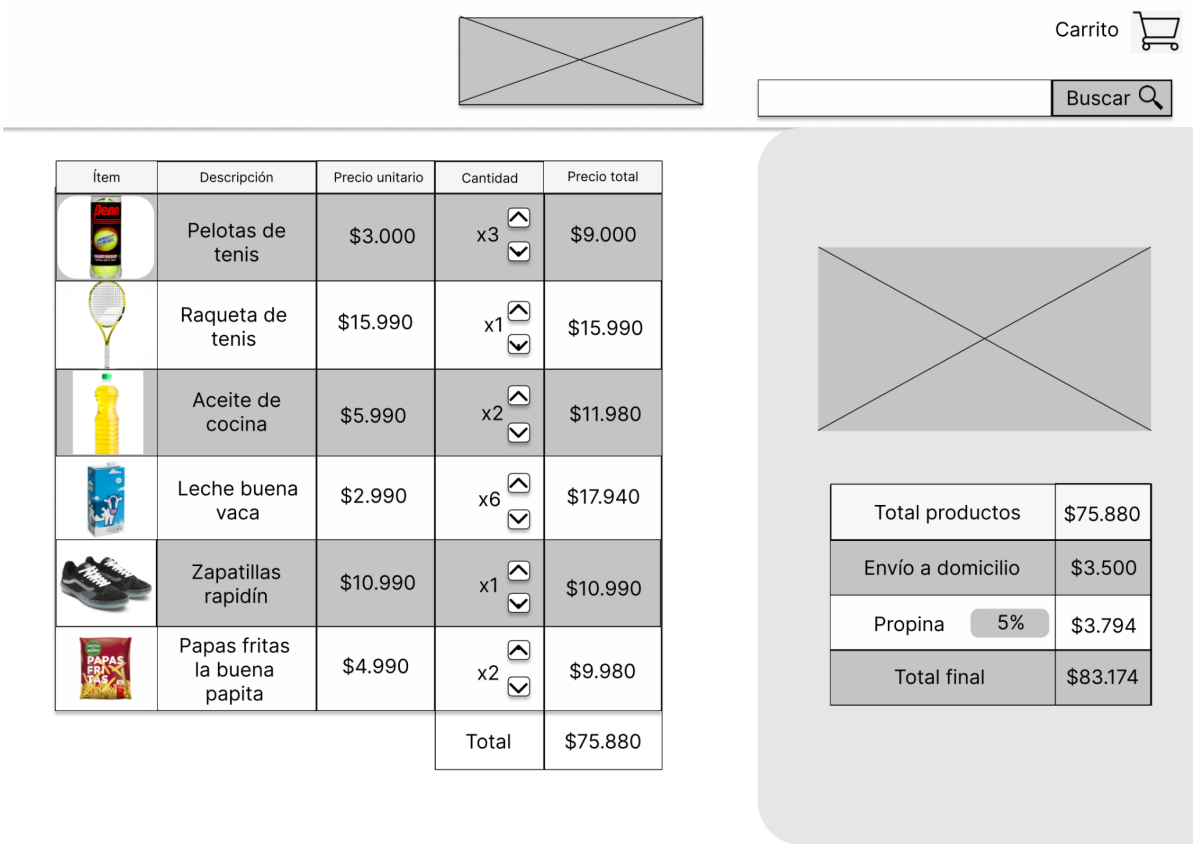

Figura 10: Carro de compras

Análisis

Durante el semestre, en el proyecto de ecommerce pudimos incorporar y poner en práctica varios de los contenidos teóricos vistos en clases. Nos basamos en los principios SOLID para la modelación de nuestro frontend y especialmente backend.

A la hora de realizar los diagramas de Entidad/Relación nos enfocamos en respetar el principio de responsabilidad única de cada modelo (S), también, al implementar los modelos creamos entidades abiertas a extensión y cerradas a modificaciones mayores (O), permitiendo que se agreguen columnas a las tablas o métodos a las clases de ser necesario en futuras entregas y tratamos de prevenir crear columnas que no se utilizarían y clases que puedan ser desechadas o que sus responsabilidades sean modificables en el futuro. Continuando con los principios, por la naturaleza sencilla del proyecto, no utilizamos mucha herencia de modelos y mantuvimos pequeño el número de clases, lo que aseguró que no contaramos con muchas oportunidades de violaciones al principio de sustitución de Liskov (L) y finalmente, el principio de inversión de dependencias (D) lo logramos entender y visualizar en las implementaciones internas del framework Ruby on Rails, que para la creación y relaciones de modelos usa dependencias a estructuras abstractas (llamadas *associations)* en lugar de hacer clases o funciones dependientes de otras estructuras concretas, por lo que al seguir las convenciones de la tecnología, evitamos crear este tipo de dependencias poco convenientes y ser consistentes con los principios de diseño de software.

Con respecto al code review, para nuestro proyecto creamos *pull requests* por cada nueva *feature.* Para poder hacer merge a la rama *development*, un compañero que no haya participado en la programación de la *pull request* debía revisar el código, verificar que no se hayan implementado malas prácticas, buscar *code smells* y probar en local los nuevos cambios para verificar que todo esté funcionando correctamente. Finalmente, cuando no había más comentarios y se resolvían los conflictos de la rama, el compañero que revisó aprobaba la *pull request* y hacía el *merge* a *development*.

Se realizó *refactor* una sola vez para la entrega 3, ya que en esta ocasión se debió juntar el código de los tres participantes por primera vez. Debido a las diferentes formas de implementar y organizar componentes en React, se decidió por un estilo usado por alguien en particular y los demás miembros realizaron el *refactor* respectivo a su parte del código, quedando así todo el proyecto con un mismo estilo. Debido a esto se necesito una reorganización de la forma de ordenar los archivos de su proyecto, también hubo que cambiar la forma de exportar un componente, manejar los props de este e incluso la forma de definir el componente en sí.

Respecto al *testing*, solamente se crearon test para el *backend*, limitándose a los modelos y controladores. La herramienta utilizada para el *testing* fue Rspec, FactoryBot y Simplecov, gemas de Ruby on Rails. La primera se utilizó para crear los test, la segunda para crear modelos falsos para poder realizar los tests correctamente y la tercera se usó para medir el porcentaje de coverage de los test. Se realizaron tests unitarios para algunas partes pequeñas del proyecto, dando entre uno y dos tests por *endpoint* debido a los distintos caminos que podía tomar el código. Este proceso dio como resultado un *coverage* mayor a 70%, superando el requerimiento del ramo.

Por último, cabe mencionar que las cosas que tuvimos en mente a la hora de realizar code reviews en los repositorios de backend y frontend fueron ciertos *code smells,* sobre todo del tipo *Bloaters* y *Dispensables*, donde se logró evitar varios de estos a lo largo del proyecto.

Conclusiones

En conclusión, el proyecto nos ayudó a poder trabajar en un ambiente pseudo-real en el desarrollo de las buenas prácticas SOLID, el code review y el desarrollo de software colaborativo.

El hecho que la E2 hubiese sido individual y la E3 haya sido reunir lo realizado por cada integrante del equipo, nos obligó a utilizar buenas prácticas en la programación ya que debíamos asegurar la limpieza y orden del código a partir de la E3, por lo que gran parte de esa entrega fue *refactorear* y definir reglas de estilos para ser consistentes en toda la aplicación, más que dedicarnos a implementar nuevas funcionalidades a la plataforma.

Algunos aspectos positivos que como equipos podemos concluir fueron que encontramos mucha productividad en el code review, ya que fue muy completo siempre y con el nivel de pausa necesario para mantener el código limpio, ya que constantemente revisamos de manera minuciosa el código y se probaba siempre. También seguimos un orden predefinido en las pull request para que sea más entendible por el resto del equipo lo realizado en esa rama.

En segundo lugar, el refactor para adaptar el código de la entrega 2 a las entrega 3 fue muy positivo, ya que logramos adaptarnos de manera rápida y fácil a la implementación de uno de nosotros para que todos usáramos la misma estructura y convenciones y así seguir el mismo orden todos.

En tercer lugar, los tests fueron un punto alto en nuestro proyecto. Esto se debe a que se cumplió más de lo mínimo solicitado en el proyecto, intentando usar buenas prácticas cómo crear modelos en una base de datos especial para testing, en modo *sandbox,* usando una gema especializada de rails. A partir de ese momento siempre se corrían los test antes de crear una pull request para verificar que no se haya roto nada, simulando la implementación de un *pipeline* de integración contínua.

Como aspectos negativos, creemos que pudimos haber priorizado en otro orden la implementación de la aplicación, ya que el tema del registro de usuarios y login es un tema que se usa transversalmente en la aplicación, y fue lo último que implementamos. Si tuviéramos que implementar de nuevo el proyecto, pondríamos el enfoque en comenzar el desarrollo haciendo las funcionalidades de los usuarios, autenticación y registro.

Otro aspecto negativo fue que no aplicamos ningún patrón de diseño conscientemente, durante el desarrollo del proyecto nunca pensamos en buscar algún patrón para la realización de alguna funcionalidad o estructura, quizás por ajustarnos a la pauta y lo solicitado en el enunciado.

Finalmente, como grupo consideramos que el proyecto tuvo un impacto positivo para nuestro aprendizaje sobre las buenas prácticas de desarrollo de software, ya que pudimos poner en práctica funciones de *testing*, *refactoring*, aplicar los principios SOLID, entre los mencionados en las secciones anteriores.