FINNEGANS

Frontend de Informe de Descuentos Totales por cliente, servicio y fecha

Sprint deadline	29/11/2023
Tarjeta	SCRUM-53
Responsable	Martín Paez

Tabla 1 Detalle de la tarjeta correspondiente en Jira

1. Objetivos. Contenido de la tarjeta

- Implementar un Hook que se comunique con la API para obtener la información de los descuentos totales de cada pedido discriminados por cliente, servicio y fecha.
- Implementar un Slice para mantener el estado interno del componente que administra dicha información obtenida de la API.
- Analizar la plantilla (excel) brindada por los tutores para determinar el modo en que se va a mostrar la información.
- Implementar la vista en formato tabla con un mecanismo que permita agrupar los datos por cliente y por servicio.

2. Dependencias

Esta tarjeta tiene dependencia con la tarjeta “SCRUM-52 Backend de Informe de Descuentos Totales por cliente, servicio y fecha”. Se colaboró con dicha implementación para

agilizar su finalización. Además, se coordinó en equipo, antes de comenzar a implementar esta tarjeta, lo que debería solicitar y retornar la API.

3. Entregable

- Sección en la aplicación web que muestre al usuario un informe acerca de los descuentos totales de cada pedido discriminados por cliente, servicio y fecha.
- Mecanismo que permita renderizar agrupaciones de registros de tablas para ser utilizado en los demás informes.

1. Procedimientos

1.1. Dependencia

Se acordó con el responsable de la tarjeta SCRUM-52 lo respectivo a como comunicarse con la API. Además, se acordó el modo en que los datos iban a estar ordenados, aprovechando así el poder del motor de base de datos. Por último, a raíz del análisis de la plantilla de informe entregada por los tutores, se trabajó en equipo con el responsable de la mencionada tarjeta para lograr unificar directamente en la base de datos los nombres de las personas con los nombres de las empresas en una única columna llamada Clientes.

1.2. Arquitectura

Esta fue la segunda vez que tomé una tarjeta del frontend, con lo cual tenía una idea mucho más clara de como estaba organizada la arquitectura. Esto agilizó mucho el trabajo, comprendía que partes del código debía modificar ya que conocía las responsabilidades asignadas a cada abstracción y en consecuencia era fácil leer el código que implementaron los demás miembros del equipo durante todo el tercer Sprint y la primera mitad del cuarto.

1.3. Metodología

Debido a que estaba bien implementada la sección de informes para el momento en que se comenzó esta tarjeta, fue bastante mecánico agregar una nueva funcionalidad.

Se sectorizó el código preexiste y se crearon los nuevos archivos donde se iba a agregar la nueva implementación. A continuación, se realizaron todas las configuraciones necesarias para vincularlos al resto de la aplicación. Esto último implicó agregar configuración a tres index.js, al store.js y al AppRouter.

Una vez comprendido cuales eran los archivos en los que se iba a trabajar y habiendo entendido el flujo de la ejecución de las instrucciones y mecanismos para almacenar y compartir información, se procedió a implementar a fondo cada una de estas piezas de código.

Al finalizar se terminaron implementando un Hook para comunicarse con la API, un Slice para mantener el estado interno de la información fruto de dicha comunicación y el Component encargado de renderizar el informe.

1.4. Grupos de registros

Había que mostrar la información agrupada por cliente y por el servicio que generó cada descuento. Esto se puede simplificar a la acción de evitar repetir nombres para ciertos campos.

Primero, se creo una tabla, y, a la hora de renderizar cada registro de la misma, se agregó una lógica de estados que permite evitar repetir nombres consecutivos iguales.

Para la primera columna este mecanismo es simple, pero para el segundo campo por el cual se agrupa, se suma la complejidad de que cada nueva agrupación se debe realizar contemplando únicamente aquellos registros que fueron agrupados previamente.

Por dar un ejemplo, si se tiene que el primer grupo esta formado por cuatro registros del mismo cliente y le sigue a continuación una agrupación de tres registros para otro cliente, puede darse el caso en que hallan nombres repetidos para servicios a agrupar para los últimos registros del primer cliente y los primeros del segundo cliente. En tal caso, dichos registro no deben agruparse entre si, si no que se deben generar dos grupos, uno por cada cliente.

Finalmente, hay que aclarar, que para que este mecanismo funcione correctamente, es necesario tener los registros agrupado previamente, o sea, ordenados de tal modo que todos

los registros del mismo cliente se listen consecutivos, y lo mismo, para la segunda agrupación, ósea, la de los servicios. Esto se decidió que era conveniente que se realice en el backend, ya que se podía aprovechar la potencia de la base de datos para realizar ese procesamiento de la información.

4. Resultados

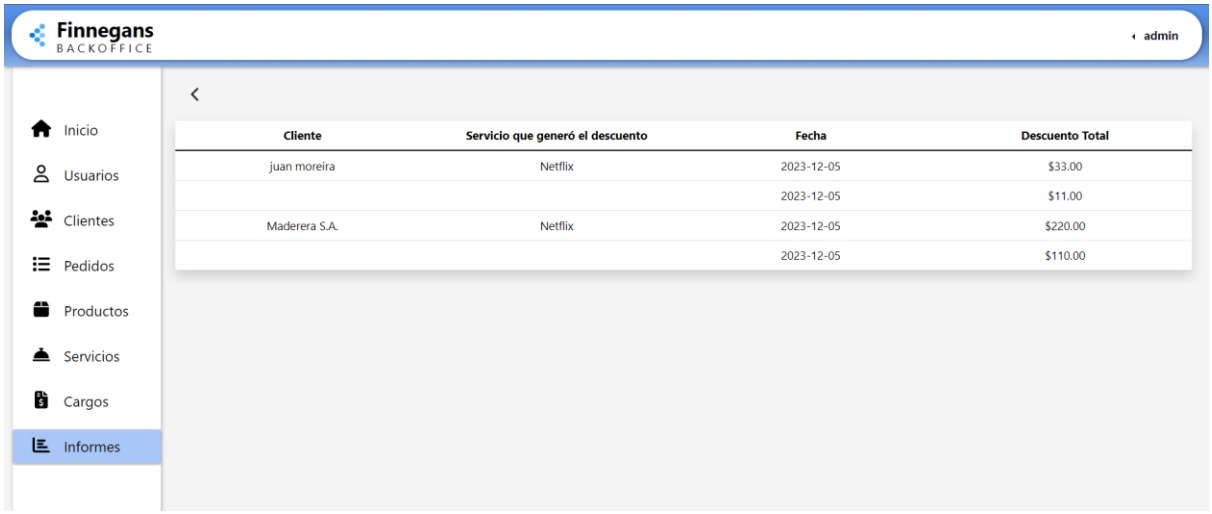

Imagen 1. Informe de descuentos totales. Se observan dos grupos para la primera columna, ya que hay dos clientes. A su vez, hay dos grupos de Servicios, uno para Juan Moreira y el otro para Maderera S.A. (el algoritmo supo diferenciarlos a pesar de que eran cuatro apariciones consecutivas de la palabra Netflix).

2. Conclusión

Siendo alguien más orientado al backend y que no conocía en absoluto React, esta segunda vez que realicé tareas de frontend noté lo mucho que aprendí en el segundo Sprint. Me pude desenvolver sin inconvenientes de un modo fluido durante el desarrollo. Incluso abordar tareas muy interesantes como la de obtener las agrupaciones en las tablas, que me permitió poner aprueba lo que aprendí acerca del modo en que React renderiza la información.