Backend de Asignación de servicios a clientes

Sprint deadline	17/11/2023
Tarjeta	SCRUM-35
Responsable	Leiva, Marcos Joel

Tabla 1 Detalle de la tarjeta correspondiente en Jira

1. Objetivos. Contenido de la tarjeta:

- Controller : Definir API.
- Model :
    - Definir constraints.
    - Definir DTO para request y response.
- Service :
    - Validación de información recibida.
    - Validaciones previas a interactuar con la BD.
    - Lógica de negocio.
- IRepository.

2. Procedimientos

2.1. Estructura nueva

La estructura sigue siendo la misma que las demás entidades.

- /controller
    - SubscriptionController
- /model
    - /client
        - /dto
            - SubscriptionRequest
            - SubscriptionResponse
        - Subscription
        - SubscriptionFactory
- /repository

        - SubscriptionRepository
    - /service
        - /interface
            - SubscriptionService
        - SubscriptionServiceImpl

3. Resultados

La aplicación ahora cuenta con un apartado detallado de las suscripciones que tiene un cliente.

3.1. Especificaciones de uso

3.1.1. Lista de Suscripciones de un cliente.

http://localhost:8080/list/{clientId}/subscriptions

- En el parámetro “clientId” de la petición debe viajar el id del cliente. El controller va a llamar un método del service de cliente que se encarga de buscar al cliente, en caso de encontrarlo va retornar la lista de suscripciones que tiene el cliente, sin importar si están habilitados o no.

```java
public List<SubscriptionResponse> getClientSubscriptions(Integer clientId) {

    Client client = validator.validateSoftDeletableEntityExistence(clientId, clientRepository);

    List<SubscriptionResponse> subscriptionsResponses = new ArrayList<>();

    for (Subscription subscriptions : client.getClientSubscriptions()) {
        subscriptionsResponses.add(subscriptionFactory.createResponse(subscriptions));
    }

    return subscriptionsResponses;
}
```

Imagen 1 Detalle del código que obtiene suscripciones de un cliente

- En cuanto al controller, el front sólo necesitará las suscripciones activas, por lo que habrá un filtro.

```java
public ResponseEntity<List<SubscriptionResponse>> getClientSubscriptions(@PathVariable int clientId) {
    // clientService.getClientSubscriptions(clientId);
    List<SubscriptionResponse> subscription = new ArrayList<>();
    for (SubscriptionResponse subscriptionResponse : clientService.getClientSubscriptions(clientId)) {
        if (subscriptionResponse.getEnabled()) {
            subscription.add(subscriptionResponse);
        }
    }
    return ResponseEntity.ok(subscription);
}
```

Imagen 2 Detalle del código que obtiene suscripciones Activas de un cliente

3.1.2. Registro de una Suscripción.

- Para el registro de una suscripción, se usará al momento de pedir una orden con un servicio, para ello se debe llamar al método “registerSubscriptions”, y por parámetros se deben pasar:
    - **Primer parámetros**: Entidad **Cliente**.
    - **Segundo parámetro**: Lista de **ServiceEntity**

```java
public void registerSubscriptions(Client client, List<ServiceEntity> services) {
    if (client == null || services == null) {
        throw new IllegalArgumentException("Client and services must not be null");
    }

    for (ServiceEntity service : services) {
        if (!clientHaveActiveSubscription(client, service)) {
            Subscription subscription = new Subscription();
            subscription.setClient(client);
            subscription.setEnabled(true);
            subscription.setService(service);

            subscriptionRepository.save(subscription);
        }else{
            System.out.println("Client already have a subscription with a service");
        }
    }
}
```

Imagen 3 Detalle del método que registra suscripciones

- Para verificar si el cliente ya tiene una suscripción activa con un servicio agregado en el pedido, se aborta el registro.

```java
public Boolean clientHaveActiveSubscription(Client client, ServiceEntity service) {
    if (client == null || service == null) {
        throw new IllegalArgumentException("Client and service must not be null");
    }

    for (Subscription sub : client.getClientSubscriptions()) {
        ServiceEntity subService = sub.getService();
        if (subService != null && subService.getId() != null && subService.getId().equals(service.getId())) {
            if(sub.isEnabled()){
                return true;
            }
        }
    }
    return false;
}
```

Imagen 3 Detalle del método que verifica si el cliente posee ya una suscripción

3.1.3. Comprobar si el cliente tiene un servicio activo

- Se trata de un método que, **por parámetro, se debe pasar una entidad "Cliente"** y su objetivo es verificar si posee un servicio activo. Retornará el primer servicio que se encuentre habilitado.

```java
public ServiceEntity getDiscountService(Client client) {
    for (Subscription subscriptions : client.getClientSubscriptions()) {
        if (subscriptions.isEnabled()) {
            return subscriptions.getService();
        } else {
            continue;
        }
    }
    return null;
}
```

4. Conclusión

La aplicación posee el registro de una subscripción a la hora de hacer la orden y es capaz de que, en la lista de clientes del frontEnd, al ver sus suscripciones, se pueda dar de baja la suscripción.