Casos de prueba para los métodos significativos de la capa de persistencia:

1. **Prueba del método `findByOwner_Id(Long ownerId)`(Escenario positivo)**
    - **Given**: Existe un cliente con ID `1` que tiene 3 cuentas asociadas.
    - **When**: Se llama al método `findByOwner_Id(1)`.
    - **Then**: Se deben retornar las 3 cuentas asociadas al cliente con ID `1`.
2. **Prueba del método `findByOwner_Id(Long ownerId)`(Escenario negativo)**
    - **Given**: No existe un cliente con ID 50
    - **When**: Se llama al método `findByOwner_Id(1)`.
    - **Then**: Se debe lanzar una excepción HTTP 404. Not Found .
3. **Prueba del método `deleteAllAccountsFromCustomer(Long id)`(Escenario positivo)**
    - **Given**: Existe un cliente con ID `2` que tiene 2 cuentas.
    - **When**: Se llama al método `deleteAllAccountsFromCustomer(2)`.
    - **Then**: Todas las cuentas asociadas al cliente con ID `2` deben ser eliminadas del repositorio.
4. **Prueba del método `deleteAllAccountsFromCustomer(Long id)`(Escenario negativo)**
    - **Given**: No existe un cliente con ID 50
    - **When**: Se llama al método `deleteAllAccountsFromCustomer(2)`.
    - **Then**: Se debe lanzar una excepción HTTP 404.Not Found.
5. **Prueba del método `withdrawFromAccounts(Long accountId, Long ownerId, int amount)` (Escenario positivo)**
    - **Given**: Una cuenta con ID `3`, perteneciente al cliente con ID `1`, con un balance de `1000`.
    - **When**: Se llama al método `withdrawFromAccounts(3, 1, 500)`.
    - **Then**: El balance de la cuenta debe ser actualizado a `500`.
6. **Prueba del método `withdrawFromAccounts(Long accountId, Long ownerId, int amount)` (Escenario negativo)**
    - **Given**: Las cuentas del cliente con ID `1` tienen un balance total de `300`.
    - **When**: Se llama al método `withdrawFromAccounts(3, 1, 500)`.
    - **Then**: Se debe lanzar una excepción `InsufficientFundsException`.

Casos de prueba para los 4 métodos más significativos de la capa de servicio:

1. **Prueba del método `findAccountByIdAndOwnerId(Long accountId, Long ownerId)`(Escenario positivo)**
    - **Given**: Existe una cuenta con ID `5` perteneciente al cliente con ID `2`.
    - **When**: Se llama al método `findAccountByIdAndOwnerId(5, 2)`.
    - **Then**: Se debe retornar la cuenta correspondiente.
2. **Prueba del método `findAccountByIdAndOwnerId(Long accountId, Long ownerId)` (Escenario negativo)**
    - **Given**: Existe una cuenta con ID `5` perteneciente al cliente con ID `2`.
    - **When**: Se llama al método `findAccountByIdAndOwnerId(5, 3)` con distinto owner.
    - **Then**: Debe retornar código http `FORBIDDEN.`
3. **Prueba del método `createAccount(AccountDTO newAccount)`(Escenario positivo)**
    - **Given**: Un `AccountDTO` válido para crear una nueva cuenta para el cliente con ID `3`.
    - **When**: Se llama al método `createAccount(newAccount)`.
    - **Then**: La nueva cuenta debe ser guardada y retornada con un ID asignado.
4. **Prueba del método `createAccount(AccountDTO newAccount)`(Escenario negativo)**
    - **Given**: Un `AccountDTO` no tiene datos válidos para crear una nueva cuenta para el cliente con ID `3`.
    - **When**: Se llama al método `createAccount(newAccount)`.
    - **Then**: La nueva cuenta no debe ser guardada y debe retornar código http `PRECONDITION_FAILED`.
5. **Prueba del método `updateAccount(AccountDTO updateAccount)`(Escenario positivo)**
    - **Given**: Existe una cuenta con ID `6` y se desea actualizar su balance a `1500`.
    - **When**: Se llama al método `updateAccount(updateAccount)` con el nuevo balance.
    - **Then**: La cuenta debe ser actualizada en el repositorio con el nuevo balance.
6. **Prueba del método `updateAccount(AccountDTO updateAccount)`(Escenario negativo)**
    - **Given**: No existe una cuenta con ID `6` en el repositorio, por lo que no se puede actualizar el balance a `1500`.
    - **When**: Se llama al método `updateAccount(updateAccount)`.
    - **Then**: Debe retornar código http `NOT_FOUND` indicando que la cuenta con ID 6 no existe.
7. **Prueba del método `deleteAccount(Long id)`(Escenario positivo)**
    - **Given**: Existe una cuenta con ID `7`.
    - **When**: Se llama al método `deleteAccount(7)`.

    - **Then**: La cuenta debe ser eliminada del repositorio.
8. **Prueba del método `deleteAccount(Long id)`(Escenario negativo)**
    - **Given**: No existe una cuenta con ID `7`.
    - **When**: Se llama al método `deleteAccount(7)`.
    - **Then**: La cuenta no existe entonces debe retornar código http `NOT_FOUND`.

Casos de prueba para los 4 métodos más significativos de la capa de vista (Controlador):

1. **Prueba del endpoint `GET /accounts/{accountId}?ownerId={ownerId}` (Escenario positivo)**
    - **Given**: Existe una cuenta con ID `8` perteneciente al cliente con ID `4`.
    - **When**: Se hace una petición GET al endpoint `/accounts/8?ownerId=4`.
    - **Then**: Se debe retornar el `AccountDTO` correspondiente con código HTTP `200 OK`.
2. **Prueba del endpoint `GET /accounts/{accountId}?ownerId={ownerId}` (Escenario negativo)**
    - **Given**: Existe una cuenta con ID `8` que no pertenece al cliente con ID `4`.
    - **When**: Se hace una petición GET al endpoint `/accounts/8?ownerId=4`.
    - **Then**: Se debe retornar un HTTP `403 FORBIDDEN`.
3. **Prueba del endpoint `POST /accounts?ownerId={ownerId}` (Escenario positivo)**
    - **Given**: Un `AccountDTO` válido para crear y el cliente con ID `5` existe.
    - **When**: Se hace una petición POST al endpoint `/accounts?ownerId=5` con el `AccountDTO` en el cuerpo.
    - **Then**: Se debe crear la cuenta y retornar el `AccountDTO` creado con código HTTP `201 Created`.
4. **Prueba del endpoint `POST /accounts?ownerId={ownerId}` (Escenario negativo)**
    - **Given**: Un `AccountDTO` no válido para crear y el cliente con ID `5` existe.
    - **When**: Se hace una petición POST al endpoint `/accounts?ownerId=5` con el `AccountDTO` en el cuerpo.
    - **Then**: Retorna el código HTTP `412 PRECONDITION_FAILED`.
5. **Prueba del endpoint `PUT /accounts/{accountId}?ownerId={ownerId}` (Escenario positivo)**
    - **Given**: Existe una cuenta con ID `9` perteneciente al cliente con ID `6`.
    - **When**: Se hace una petición PUT al endpoint `/accounts/9?ownerId=6` con los datos actualizados.

    - **Then**: La cuenta debe ser actualizada y se debe retornar el `AccountDTO` actualizado con código HTTP `202 Accepted`.
6. **Prueba del endpoint `PUT /accounts/{accountId}?ownerId={ownerId}` (Escenario negativo)**
    - **Given**: No existe una cuenta con ID `9` perteneciente al cliente con ID `6`.
    - **When**: Se hace una petición PUT al endpoint `/accounts/9?ownerId=6` con los datos actualizados.
    - **Then**: La cuenta no existe entonces debe retornar HTTP `404 NOT_FOUND`.
7. **Prueba del endpoint `DELETE /accounts/{accountId}?ownerId={ownerId}` (Escenario positivo)**
    - **Given**: Existe una cuenta con ID `10` perteneciente al cliente con ID `7`.
    - **When**: Se hace una petición DELETE al endpoint `/accounts/10?ownerId=7`.
    - **Then**: La cuenta debe ser eliminada y se debe retornar código HTTP `204 No Content`.
8. **Prueba del endpoint `DELETE /accounts/{accountId}?ownerId={ownerId}` (Escenario negativo)**
    - **Given**: No existe una cuenta con ID `10` perteneciente al cliente con ID `7`.
    - **When**: Se hace una petición DELETE al endpoint `/accounts/10?ownerId=7`.
    - **Then**: La cuenta no existe entonces debe retornar HTTP `404 NOT_FOUND`.

Casos de prueba para validar la integración entre la capa de servicio y persistencia (2 escenarios significativos):

1. **Escenario positivo: Crear y guardar una nueva cuenta**
    - **Given**: Un `AccountDTO` válido para el cliente con ID `8`.
    - **When**: Se llama al servicio `createAccount` y este interactúa con el repositorio.
    - **Then**: La cuenta debe ser persistida en la base de datos y retornada correctamente.
2. **Escenario negativo: Intentar eliminar una cuenta que no existe**
    - **Given**: Una cuenta con ID `999` que no existe en el repositorio.
    - **When**: Se llama al servicio `deleteAccount(999)`.
    - **Then**: Se debe lanzar una excepción `AccountNotFoundException`.

Casos de prueba para validar la integración entre la capa de vista y servicio (2 escenarios significativos):

1. **Escenario positivo: Consultar cuentas de un cliente existente**
    - **Given**: El cliente con ID `9` tiene 2 cuentas existentes.
    - **When**: Se hace una petición GET a `/accounts/user/9`.
    - **Then**: El controlador debe llamar al servicio correspondiente y retornar las 2 cuentas con código HTTP `200 OK`.
2. **Escenario negativo: Intentar crear una cuenta para un cliente que no existe**
    - **Given**: Un `AccountDTO` válido y un cliente con ID `999` que no existe.
    - **When**: Se hace una petición POST a `/accounts?ownerId=999` con el `AccountDTO`.
    - **Then**: El controlador debe manejar la excepción `CustomerNotFoundException` y retornar un código HTTP `404 Not Found`.