

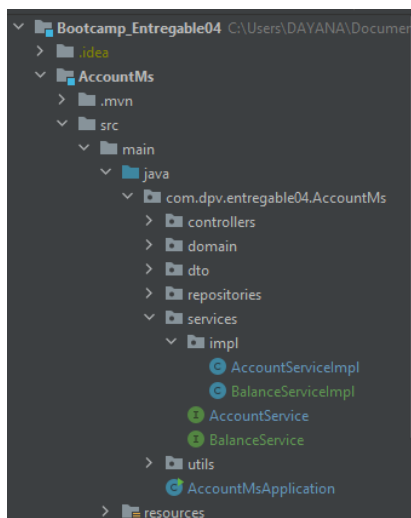
PRINCIPIO SOLID

Microservicios:

AccountMs

Principio de Responsabilidad Única

Se utiliza el Principio de Responsabilidad Única en los package de Service y en impl, creándose dos interfaces (AccountService y BalanceService) y dos clases (AccountServiceImpl y BalanceServiceImpl), en donde la clase de AccountService tiene las funcionalidades de registrar, listar y eliminar cuentas, mientras que la clase BalanceService presenta las funcionalidades de depositar y remover el saldo de las cuentas.

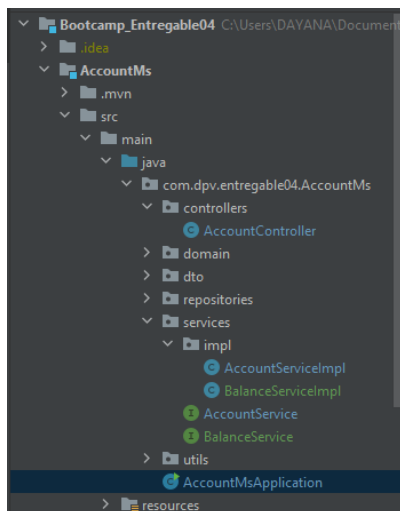


```
AccountService.java
1 package com.dpv.entregable04.AccountMs.services;
2
3 import com.dpv.entregable04.AccountMs.domain.Account;
4 import com.dpv.entregable04.AccountMs.dto.AccountRequest;
5
6 import java.util.List;
7
8 public interface AccountService {
9     Account saveAccount(AccountRequest accountRequest);
10    List<Account> listAccounts();
11    Account getAccountById(Long id);
12    List<Account> getAccountsByCustomerId(Long customerId);
13    boolean deleteAccount(Long id);
14 }
15
```

```
BalanceService.java
1 package com.dpv.entregable04.AccountMs.services;
2
3 import com.dpv.entregable04.AccountMs.domain.Account;
4
5 public interface BalanceService {
6     Account depositBalance(Long id, Double amount);
7     Account depositBalanceAccount(String accountNumber, Double amount);
8     Account removeBalance(Long id, Double amount);
9     Account removeBalanceAccount(String accountNumber, Double amount);
10 }
11
```

Principio de Inversión de Dependencias

Se utiliza el Principio de Inversión de Dependencias en el Controller AccountController, debido a que el controlador no tiene conocimiento de la implementación concreta de AccountServiceImpl, ya que se conecta con la interfaz AccountService.

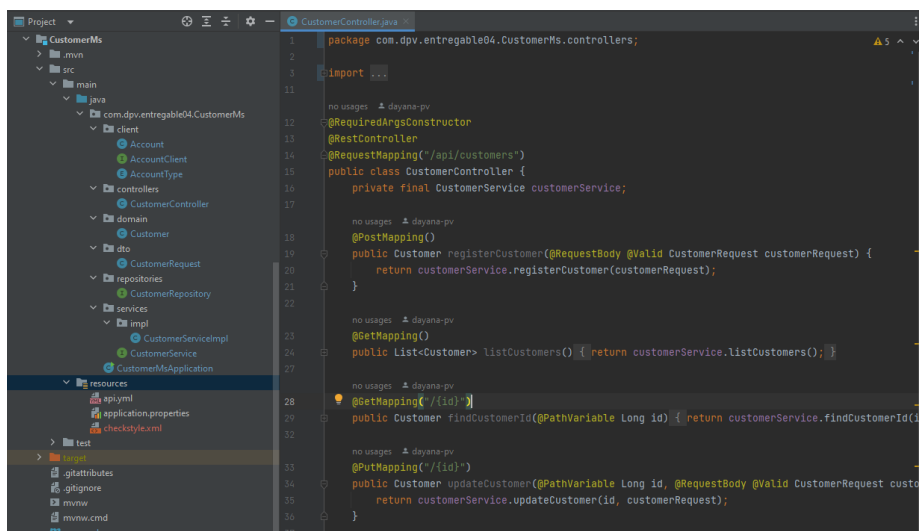


```
1 package com.dpv.entregable04.AccountMs.controllers;
2
3 import ...
4
5 no usages: A dayana-pv
6
7 @RequiredArgsConstructor
8 @RestController
9 @RequestMapping("/api/accounts")
10 public class AccountController {
11     private final AccountService accountService;
12     private final BalanceService balanceService;
13
14     no usages: A dayana-pv
15     @PostMapping()
16     public Account registerAccount(@RequestBody @Valid AccountRequest accountRequest) {
17         return accountService.saveAccount(accountRequest);
18     }
19
20     no usages: A dayana-pv
21     @GetMapping()
22     public List<Account> listAccounts() { return accountService.listAccounts(); }
23
24     no usages: A dayana-pv
25     @GetMapping("/{id}/{id}")
26     public Account getAccountById(@PathVariable Long id) { return accountService.getAccountById(id); }
27
28     no usages: A dayana-pv
29     @GetMapping("/{customer}/{customerid}")
30     public List<Account> getAccountsByCustomerId(@PathVariable Long customerId) {
31         return accountService.getAccountsByCustomerId(customerId);
32     }
33
34     no usages: new?
35     @PostMapping("/{deposit}/{id}")
36     public Account depositBalance(@PathVariable Long id, @RequestBody @Valid BalanceRequest balanceRequest) {
37         return balanceService.depositBalance(id, balanceRequest.getBalance());
38     }
39 }
```

CustomerMS

Principio de Inversión de Dependencias

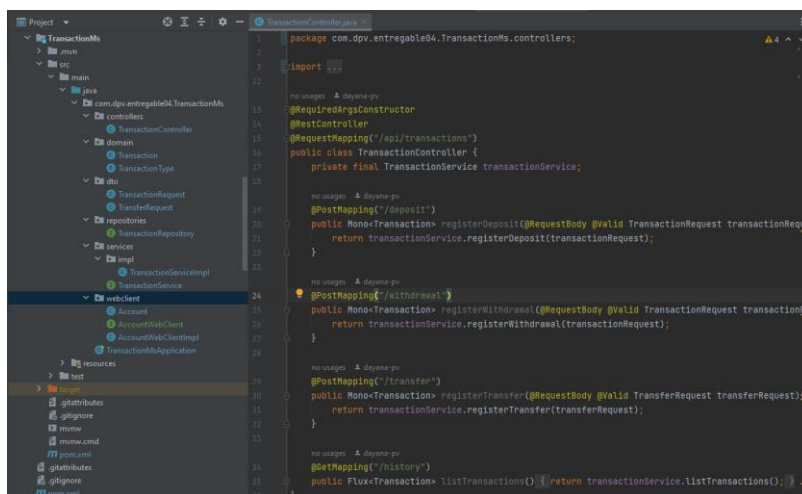
Se utiliza el Principio de Inversión de Dependencias en el Controller CustomerController, debido a que el controlador no tiene conocimiento de la implementación concreta de CustomerServiceImpl, ya que se conecta con la interfaz CustomerService.



TransactionMS

Principio de Inversión de Dependencias

Se utiliza el Principio de Inversión de Dependencias en el Controller TransactionController, debido a que el controlador no tiene conocimiento de la implementación concreta de TransactionServiceImpl, ya que se conecta con la interfaz TransactionService.



PATRONES DE DISEÑO

SINGLETON

Se utiliza el Patrón de Diseño Singleton en las clases AccountNumberGenerator y AccountServiceImpl, garantizando que AccountServiceImpl esté funcionando correctamente para obtener la instancia única de AccountNumberGenerator.

