

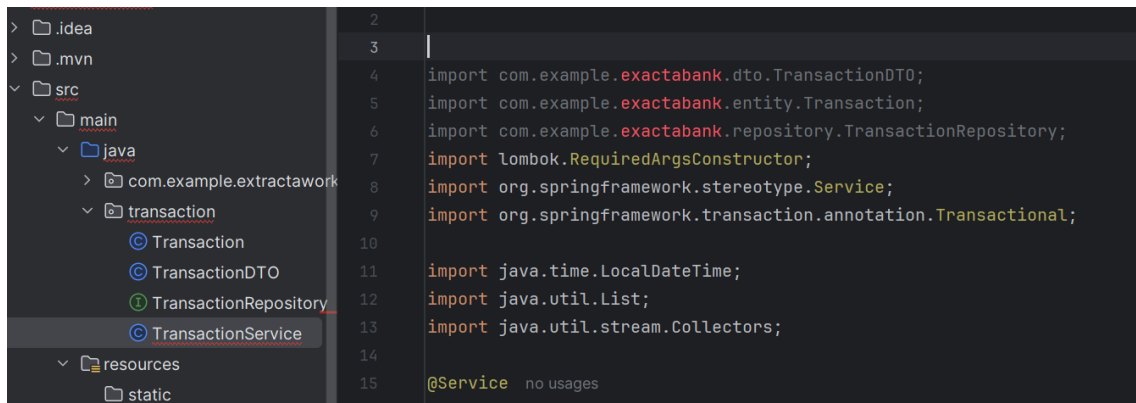
## Sumário

- 1- Decisões Arquiteturais
- 2- Configuração do Banco de dados Postgres
- 3- Build do projeto
- 4- Executando testes unitários com Junit e Mockito

Repositório Git : <https://github.com/pvsobrinho/exactaworks>

## Decisões Arquiteturais

Apliquei a arquitetura em camadas usando alguns princípios de SOLID. Por exemplo, eu agrupei todos os arquivos por domínio. Assim, é mais fácil encontrar e dar manutenção em arquivos. Em se tratando de projetos grandes com muitas entidades, isso é uma das vantagens desse modelo arquitetural. Além disso, existe uma menor necessidade de utilizar importes, já que a maioria dos arquivos condizem a aquele domínio estar dentro do mesmo pacote. Existem outras vantagens desse modelo arquitetural, mas essa é uma das vantagens nítidas desse projeto pequeno.



Na arquitetura SOLID. Além de ser orientado a domínio que facilita a organização das entidades de domínio. Precisamos importar com uma frequência muito menor. Pois os pacotes são separados por domínio e não por responsabilidade. Isso é muito útil e produtivo em projetos que existem muitas entidades. Existem outras vantagens da arquitetura SOLID. As linhas 4,5 e 6 não são necessárias nesse modelo de arquitetura.

### Motivos para Usar DTOs em vez de Entidades em Todo o Projeto

#### 1. Encapsulamento e Separação de Preocupações

- Encapsulamento de Dados:

- Isolamento da Lógica de Negócio:
- 2. Segurança e Controle de Dados**
- Controle sobre os Dados Expostos
  - Redução de Riscos de Manipulação Indevida
- 3. Flexibilidade e Evolução da API**
- Facilidade de Modificação da API
  - Versionamento de API
- 4. Desempenho e Otimização**
- Redução de Dados Transferidos
  - Transformação de Dados
- 5. Testabilidade e Manutenibilidade**
- Facilidade de Testes.
  - Manutenção Simplificada
- 6. Mapeamento e Transformação de Dados**
- Mapeamento Claro
  - Transformação de Estruturas Complexas

## Configuração do Banco de Dados Postgres

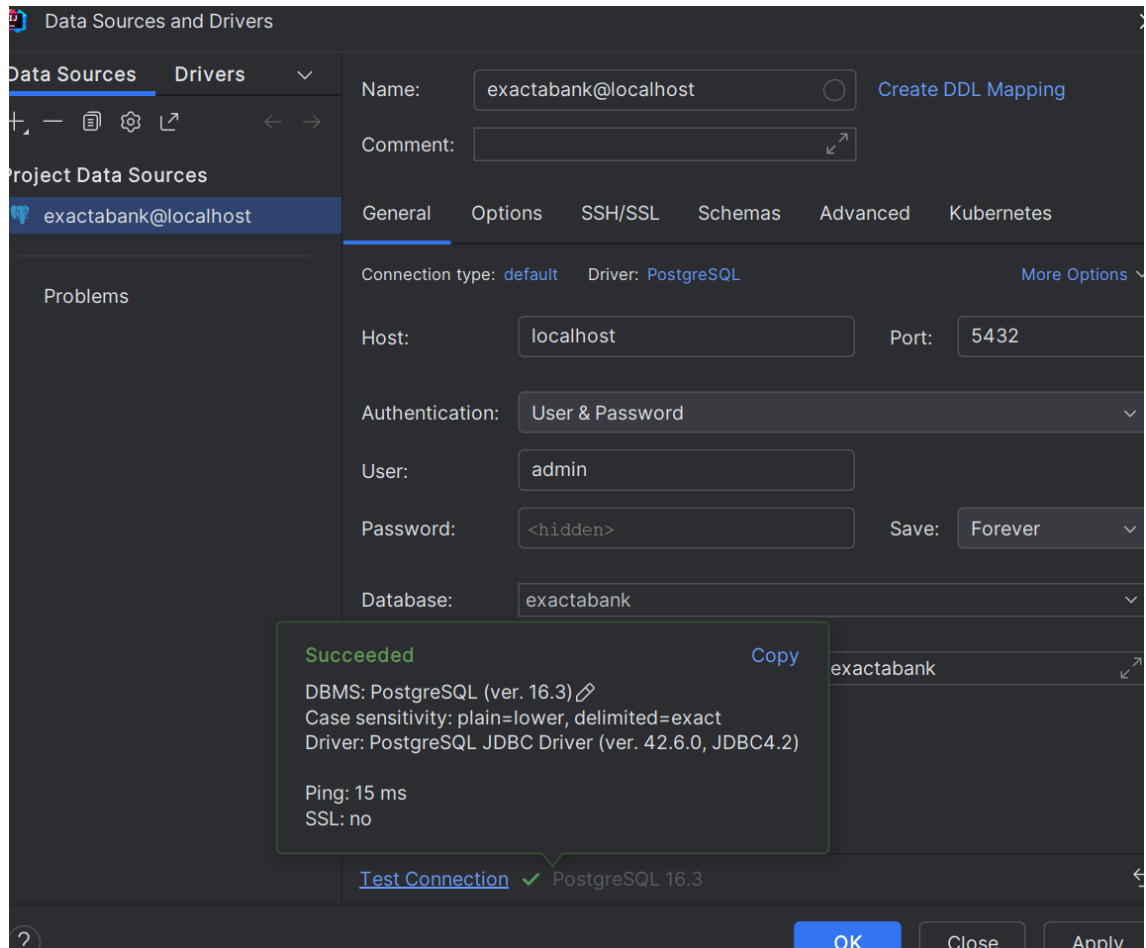
```
C:\Windows\System32>psql -U postgres
'psql' não é reconhecido como um comando interno
ou externo, um programa operável ou um arquivo em lotes.

C:\Windows\System32>cd C:\Program Files\PostgreSQL\16\bin

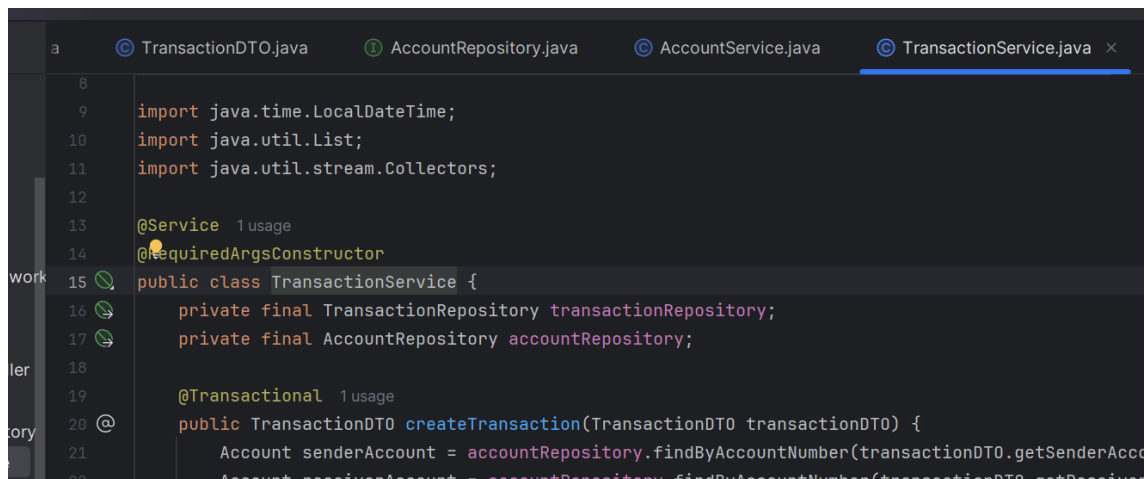
C:\Program Files\PostgreSQL\16\bin>psql -U postgres
Senha para o usuário postgres:
psql (16.3)
ADVERTÊNCIA: A página de código da console (850) difere da página de código do Windows (1252)
os caracteres de 8 bits podem não funcionar corretamente. Veja a página de
referência do psql "Notes for Windows users" para obter detalhes.
Digite "help" para obter ajuda.

postgres=# CREATE DATABASE exactabank;
CREATE DATABASE
postgres=# CREATE USER admin WITH PASSWORD 'admin';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE exactabank TO admin;
GRANT
postgres=#
```

Configurando o postgres localmente. Definido um usuário admin. Caso não tenha esta etapa é opcional.



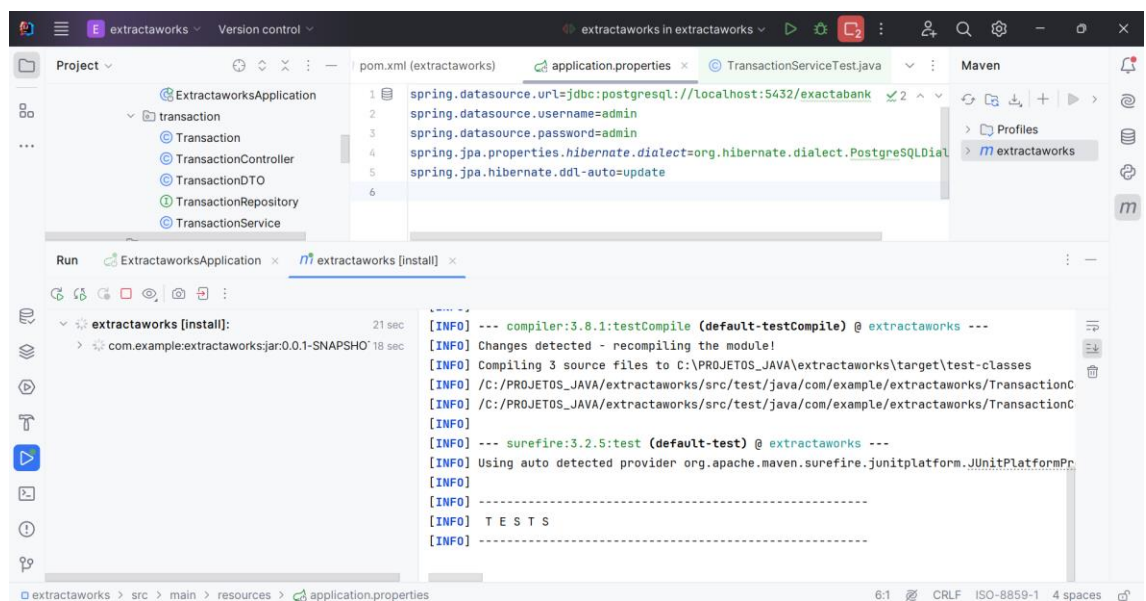
Usar `private final` em conjunto com a anotação `@RequiredArgsConstructor` é uma abordagem que segue as melhores práticas de injeção de dependências no Spring, conhecida como injeção de dependências pelo construtor. Essa abordagem tem várias vantagens sobre a anotação `@Autowired` em campos, incluindo:



```
8
9 import java.time.LocalDateTime;
10 import java.util.List;
11 import java.util.stream.Collectors;
12
13 @Service 1 usage
14 @RequiredArgsConstructor
15 public class TransactionService {
16     private final TransactionRepository transactionRepository;
17     private final AccountRepository accountRepository;
18
19     @Transactional 1 usage
20     public TransactionDTO createTransaction(TransactionDTO transactionDTO) {
21         Account senderAccount = accountRepository.findByAccountNumber(transactionDTO.getSenderAccountNumber());
22         Account receiverAccount = accountRepository.findByAccountNumber(transactionDTO.getReceiverAccountNumber());
```

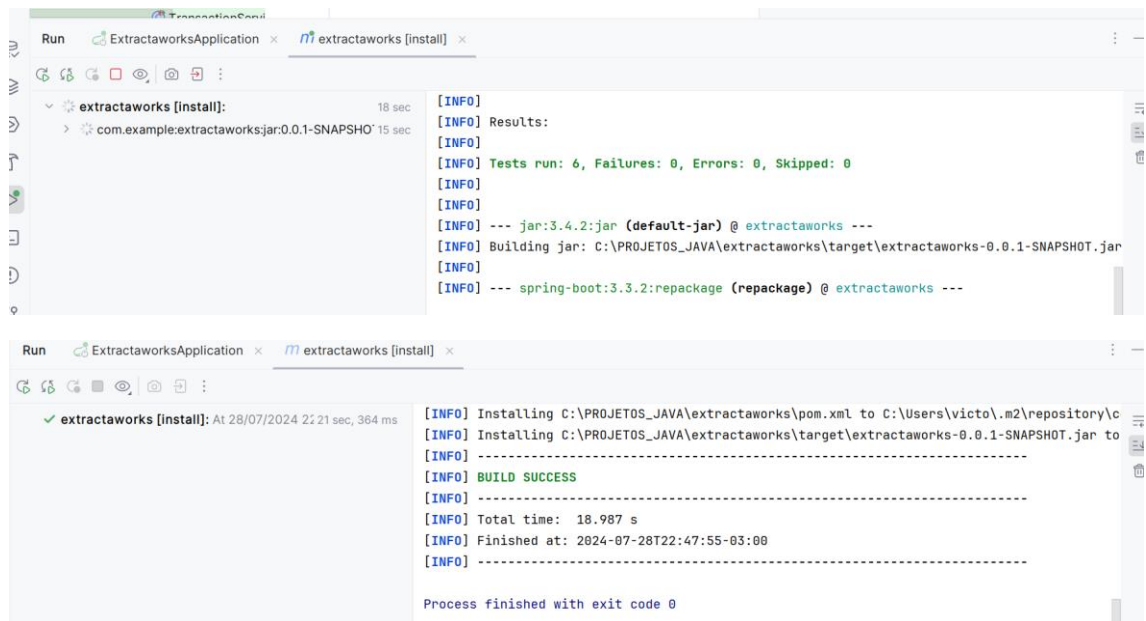
1. **Imutabilidade:** Usar final nos campos torna claro que esses campos não devem ser alterados após a inicialização, promovendo a imutabilidade.
2. **Testabilidade:** Facilita a criação de instâncias de classes para testes, pois você pode passar dependências diretamente pelo construtor.
3. **Clareza e segurança:** Torna explícito quais são as dependências da classe, melhorando a legibilidade e a manutenção do código.

## Build do projeto



```
1 spring.datasource.url=jdbc:postgresql://localhost:5432/exactabank
2 spring.datasource.username=admin
3 spring.datasource.password=admin
4 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
5 spring.jpa.hibernate.ddl-auto=update
6
```

```
[INFO] --- compiler:3.8.1:testCompile (default-testCompile) @ extractaworks ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 3 source files to C:\PROJETOS_JAVA\extractaworks\target\test-classes
[INFO] /C:/PROJETOS_JAVA/extractaworks/src/test/java/com/example/extractaworks/TransactionC
[INFO] /C:/PROJETOS_JAVA/extractaworks/src/test/java/com/example/extractaworks/TransactionC
[INFO] --- surefire:3.2.5:test (default-test) @ extractaworks ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformPr
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
```



```
Run ExtractaworksApplication x extractaworks [install] x
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.4.2:jar (default-jar) @ extractaworks ---
[INFO] Building jar: C:\PROJETOS_JAVA\extractaworks\target\extractaworks-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot:3.3.2:repackage (repackage) @ extractaworks ---
[INFO]
[INFO] Installing C:\PROJETOS_JAVA\extractaworks\pom.xml to C:\Users\victo\.m2\repository\c
[INFO] Installing C:\PROJETOS_JAVA\extractaworks\target\extractaworks-0.0.1-SNAPSHOT.jar to
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 18.987 s
[INFO] Finished at: 2024-07-28T22:47:55-03:00
[INFO]
Process finished with exit code 0
```

Execute o comando: `Mvn- install` ao abrir o projeto. Se houver algum tipo de erro nesta etapa, provavelmente pode ser a codificação. Este projeto está configurado para ser executado em UTF-8. Verifique a configuração da sua IDE.

## Executando os testes unitários com Junit e Mockito

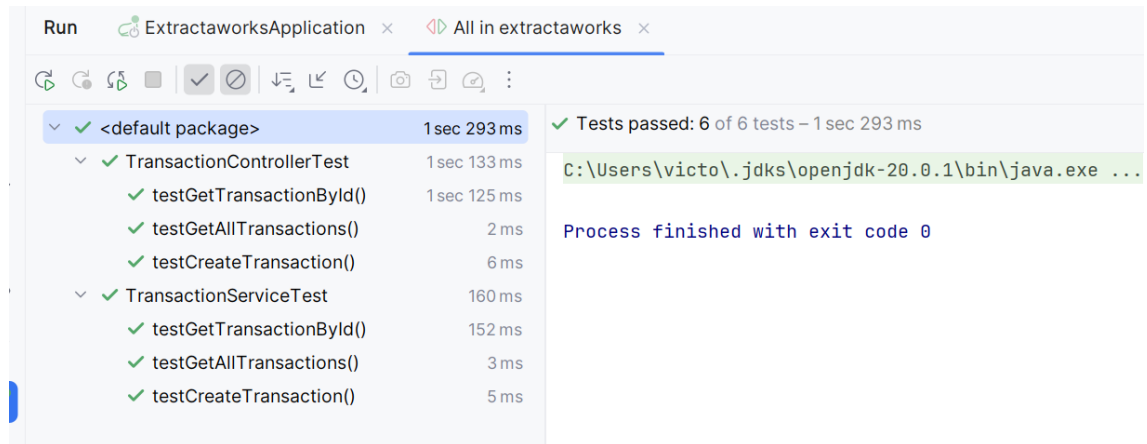
Execute o comando- `mvn test`.

Para executar com log detalhado:

`mvn -X test`

?

Criei testes unitários referentes à entidade de transação. Pois não faz muito sentido criar testes unitários da entidade de conta, que serve, neste caso, apenas como identificador das transações. A operação que é realizada aqui é de transação, e não de conta. Portanto, como é solicitado neste desafio, é possível fazer diversos tipos de transação, havendo o identificador e o nome da transação, se ela é de recarga ou de transferência bancária. A diferença é apenas no nome e no identificador. De toda forma, o valor será creditado e deduzido da conta.



## Documentação e Swagger

Considerando que o serviço foi iniciado na porta 8080. Acesse no seu navegador para acessar o Swagger e documentação: **<http://localhost:8080/swagger-ui/index.html>**