Testes camada de Serviço

Laboratório de Programação

Objetivo

- implementar testes de unidade de regras de negócio
- avaliar retornsos e tratamentos

Começando: UsuarioService

- Testes de restrições:
    - deveGerarErroAoTentarSalvarSemNome
    - deveGerarErroAoTentarSalvarSemSenha
    - deveGerarErroAoTentarSalvarSemEmail
    - deveGerarErroSalvarUsuarioComMesmoEmail
    - deveGerarErroEmailaoTentarAutenticarUsuario
    - deveGerarErroSenhalaoTentarAutenticarUsuario
    - deveObterSaldosDeInvestimentosVazioQuandoNaoHouverInvestimento
- Teste de funcionalidades:
    - deveSalvarUsuario
    - deveAutenticarUsuario
    - deveObterSaldosDeInvestimentos

Também houveram mudanças em UsuarioRepository

- Se método estiver usando anotações do SpringData
  - não é necessário testar a princípio
- Teste de funcionalidade:
  - deveVerificarSaldoInvestimentos

Vamos logo pelo `UsuarioRepository`

Método:

```
@Query("select new com.labprog.patrimonio.model.dto.InvestimentoSal
        "from Posicao p join p.investimento i "+
        "where i.usuario = :usuario " +
        "group by i")
List<InvestimentoSaldo> obterSaldosInvestimentos(@Param("usuario")
```

- Cenário: inserir uns dados de usuário, investimento e posição
- ação: chamar a consulta
- check: verificar se o valor retornado é igual ao esperado

Vamos logo pelo `UsuarioRepository`

```
@Test
public void deveVerificarSaldoInvestimentos() {
    //cenário
    Usuario user = Usuario.builder().nome("Teste")
            .email("teste@teste.com")
            .senha("123").build();
    Usuario salvo = repository.save(user);

    Investimento inv1 = Investimento.builder()
                .nome("Primeiro")
                .usuario(salvo).build();

    Investimento inv1Salvo = investimentoRep.save(inv1);

    Posicao p1 = Posicao.builder()
                .valor(500.0)
                .data(new Date())
                .investimento(inv1Salvo).build();
    Posicao p2 = Posicao.builder().valor(500.0).data(new Date()).in

    Posicao p1salvo = posicaoRep.save(p1);
    Posicao p2salvo = posicaoRep.save(p2);
```

Vamos logo pelo `UsuarioRepository`

```
  //ação
  List<InvestimentoSaldo> res = repository.
                    obterSaldosInvestimentos(salvo);

  //verificação
  Assertions.assertNotNull(res);
  Assertions.assertEquals(1000.0,res.get(0).valor);

  //retorno
  posicaoRep.delete(p1salvo);
  posicaoRep.delete(p2salvo);

  investimentoRep.delete(inv1Salvo);

  repository.delete(salvo);
}
```

Agora ao `UsuarioService`

- Começando pelas definições:

```
@ExtendWith(SpringExtension.class)
@SpringBootTest
@ActiveProfiles("test")
public class UsuarioServiceTest {

    @Autowired
    UsuarioService service;

    @Autowired
    UsuarioRepository repository;

    @Autowired
    InvestimentoRepository investimentoRep;

    @Autowired
    PosicaoRepository posicaoRep;
```

Agora ao `UsuarioService`

- e agora as restrições

```
@Test
public void deveGerarErroAoTentarSalvarSemNome() {
    Usuario usr = Usuario.builder()
                                .email("teste")
                                .senha("123")
                                .build();
    Assertions.assertThrows(RegraNegocioRunTime.class,
                                () -> service.salvar(usr),
                                "Nome do usuário deve ser informa
}
```

- É esperado uma exeção quando tentar salvar usuário sem nome
  - usamos `assertThrows`
  - que espera a classe `RegraNegocioRunTime`
  - com a mensagem `Nome do usuário deve ser informado`

Agora ao `UsuarioService`

- continuando

```
@Test
public void deveGerarErroAoTentarSalvarSemSenha() {
    Usuario usr = Usuario.builder()
                         .nome("teste")
                         .email("teste")
                         .build();
    Assertions.assertThrows(
          RegraNegocioRunTime.class,
                         () -> service.salvar(usr),
                         "Usuário deve possui senha");
}

@Test
public void deveGerarErroAoTentarSalvarSemEmail() {
    Usuario usr = Usuario.builder()
                         .nome("teste")
                         .senha("teste")
                         .build();
    Assertions.assertThrows(
          RegraNegocioRunTime.class,
                         () -> service.salvar(usr),
                         "Email deve ser informado");
}
```

Agora ao `UsuarioService`

- continuando

```
@Test()
public void deveGerarErroSalvarUsuarioComMesmoEmail() {
    Usuario usuario = Usuario.builder().nome("Teste")
                                       .email("123@teste.com")
                                       .senha("123").build();

    Usuario salvo = service.salvar(usuario);
    Assertions.assertThrows(
            RegraNegocioRunTime.class,
                    () -> service.salvar(usuario));
    repository.delete(salvo);
}
```

Agora ao `UsuarioService`

- continuando

```
@Test
public void deveObterSaldosDeInvestimentosVazioQuandoNaoHouverI
Usuario usuario = Usuario.builder().nome("Teste")
                                   .email("123@teste.com")
                                   .senha("123").build();
Usuario salvo = repository.save(usuario);

List<InvestimentoSaldo> invs = service.obterSaldos(salvo);

Assertions.assertEquals(0, invs.size());
repository.delete(salvo);
}
```

Agora ao `UsuarioService`

- continuando

```
@Test
public void deveGerarErroEmailaoTentarAutenticarUsuario() {

  Assertions.assertThrows(
      RegraNegocioRunTime.class,
            () -> service.efetuarLogin("não existe", "123")
            ,"Erro de autenticação. Email informado não encontrado"
}

@Test
public void deveGerarErroSenhalaoTentarAutenticarUsuario() {

  Usuario usuario = Usuario.builder().nome("Teste")
                           .email("123@teste.com")
                           .senha("123").build();
  Usuario salvo = repository.save(usuario);
  Assertions.assertThrows(
      RegraNegocioRunTime.class,
             () -> service.efetuarLogin("123@teste.com", "aaa")
             ,"Erro de autenticação. Senha inválida");
  repository.delete(salvo);
}
```

Agora ao `UsuarioService`

- Funcionalidades: salvar usuario

```
@Test
public void deveSalvarUsuario() {
    Usuario usuario = Usuario.builder()
        .nome("Teste")
        .email("123@teste.com")
        .senha("123").build();

    Usuario salvo = service.salvar(usuario);

    Assertions.assertNotNull(salvo);
    Assertions.assertNotNull(salvo.getId());

    repository.delete(salvo);
}
```

Agora ao `UsuarioService`

- Funcionalidades: autenticar usuario

```
@Test
public void deveAutenticarUsuario() {

    Usuario usuario = Usuario.builder().nome("Teste")
                                       .email("123@teste.com")
                                       .senha("123").build();
    Usuario salvo = repository.save(usuario);
    boolean resposta = service.
            efetuarLogin("123@teste.com", "123");
    Assertions.assertTrue(resposta);

    repository.delete(salvo);
}
```

Agora ao `UsuarioService`

- Funcionalidades: obter saldos
  - primeiro criar todo o cenário
  - depois partir para o teste

```
@Test
public void deveObterSaldosDeInvestimentos() {
  //cenário
  Usuario usuario = Usuario.builder()
                  .nome("Teste")
                  .email("123@teste.com")
                  .senha("123").build();
  Usuario salvo = repository.save(usuario);

  Investimento investimento = Investimento.builder()
                  .nome("Teste")
                  .usuario(salvo).build();
  Investimento investimentoSalvo = investimentoRep.save(investiment

  Posicao pos = Posicao.builder()
                  .valor(500.0)
                  .data(new Date())
                  .investimento(investimentoSalvo).build();
  Posicao posSalvo = posicaoRep.save(pos);
```

Agora ao `UsuarioService`

```
@Test
public void deveObterSaldosDeInvestimentos() {
  //cenário
  Usuario usuario = Usuario.builder().nome("Teste")
                                     .email("123@teste.com")
                                     .senha("123").build();
  Usuario salvo = repository.save(usuario);

  Investimento investimento = Investimento.builder().nome("Teste").
  Investimento investimentoSalvo = investimentoRep.save(investiment

  Posicao pos = Posicao.builder().valor(500.0).data(new Date()).inv
  Posicao posSalvo = posicaoRep.save(pos);
  //ação
  List<InvestimentoSaldo> invs = service.obterSaldos(salvo);

  //verificação
  Assertions.assertNotNull(invs);
  Assertions.assertEquals(1, invs.size());
  Assertions.assertEquals(500, invs.get(0).valor);

  posicaoRep.delete(posSalvo);
  investimentoRep.delete(investimentoSalvo);
  repository.delete(salvo);
}
```

Agora é com você:

Implemente os testes para InvestimentoService e PosicaoService como aprendizado