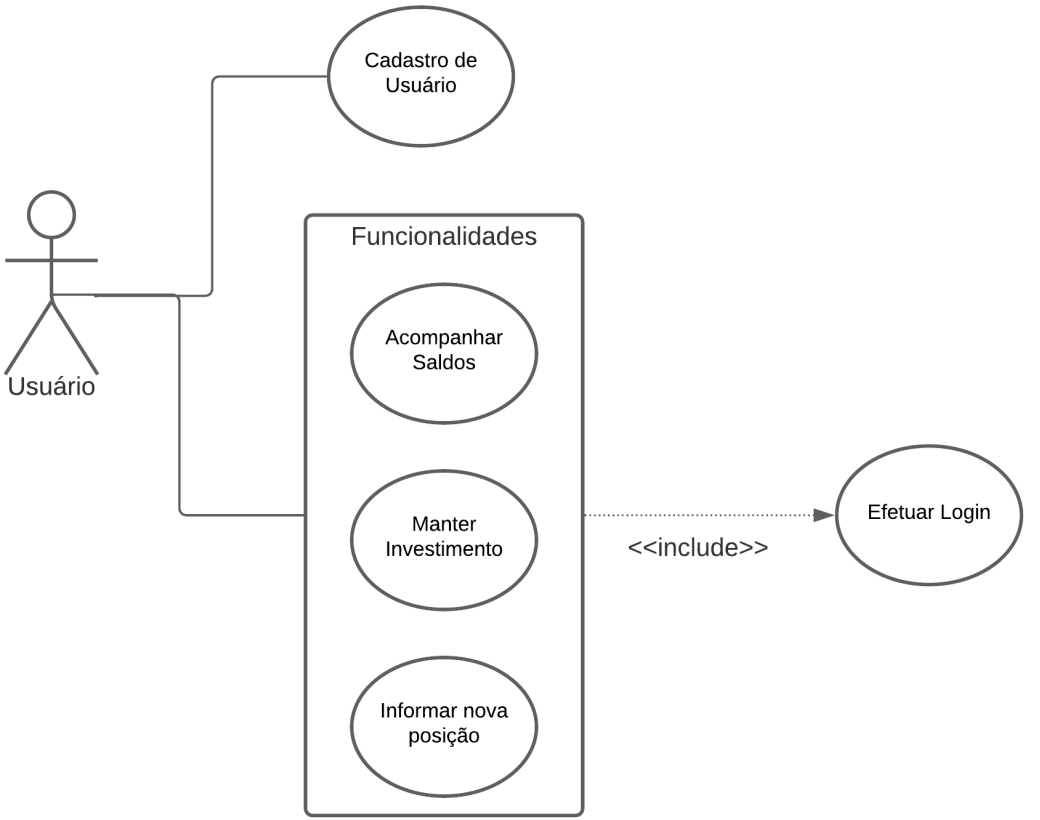
Camada Serviço

Laboratório de Programação

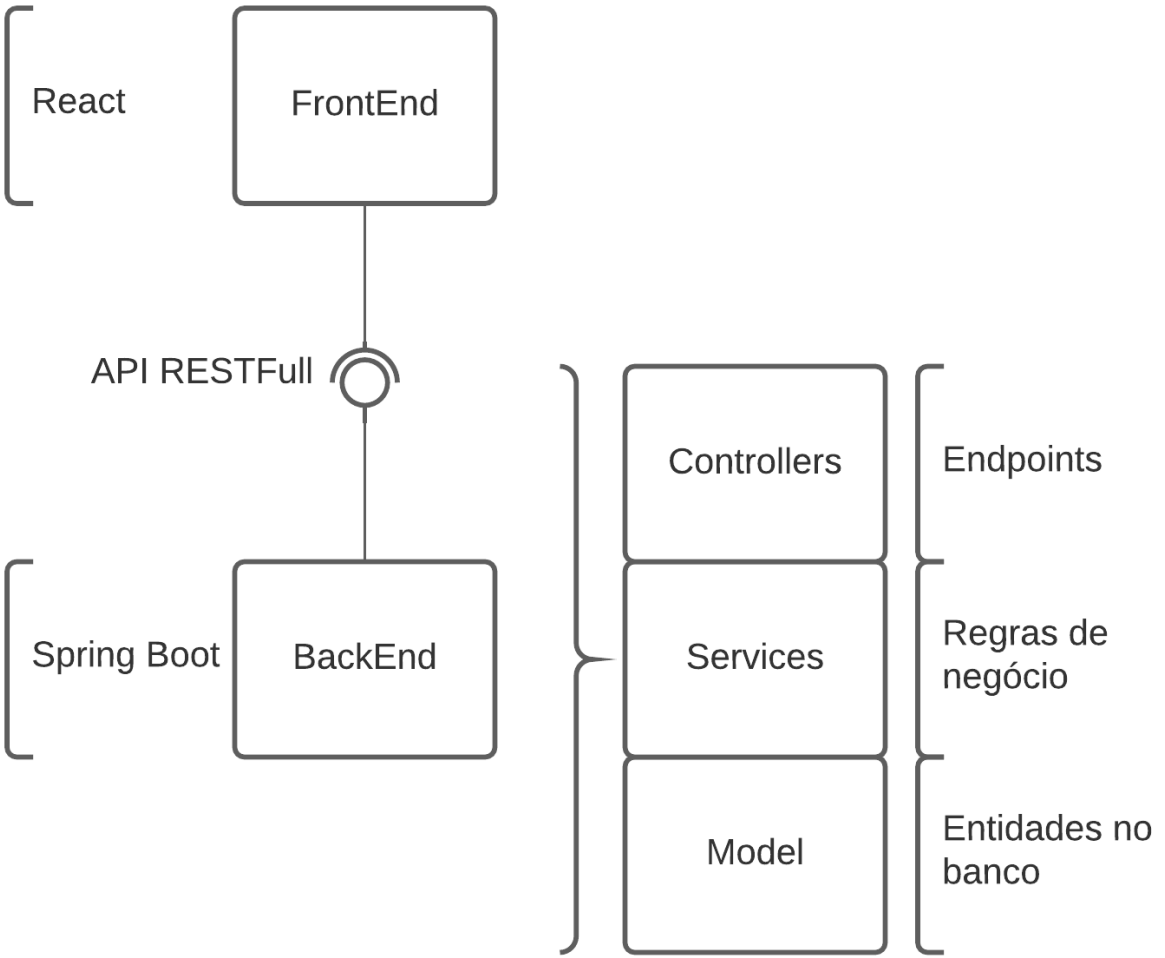
Objetivos

- demonstrar implementação das funcionalidades de regra de negócio
- dar início a modelo de exceções
- atualizar repositórios com funções especializadas

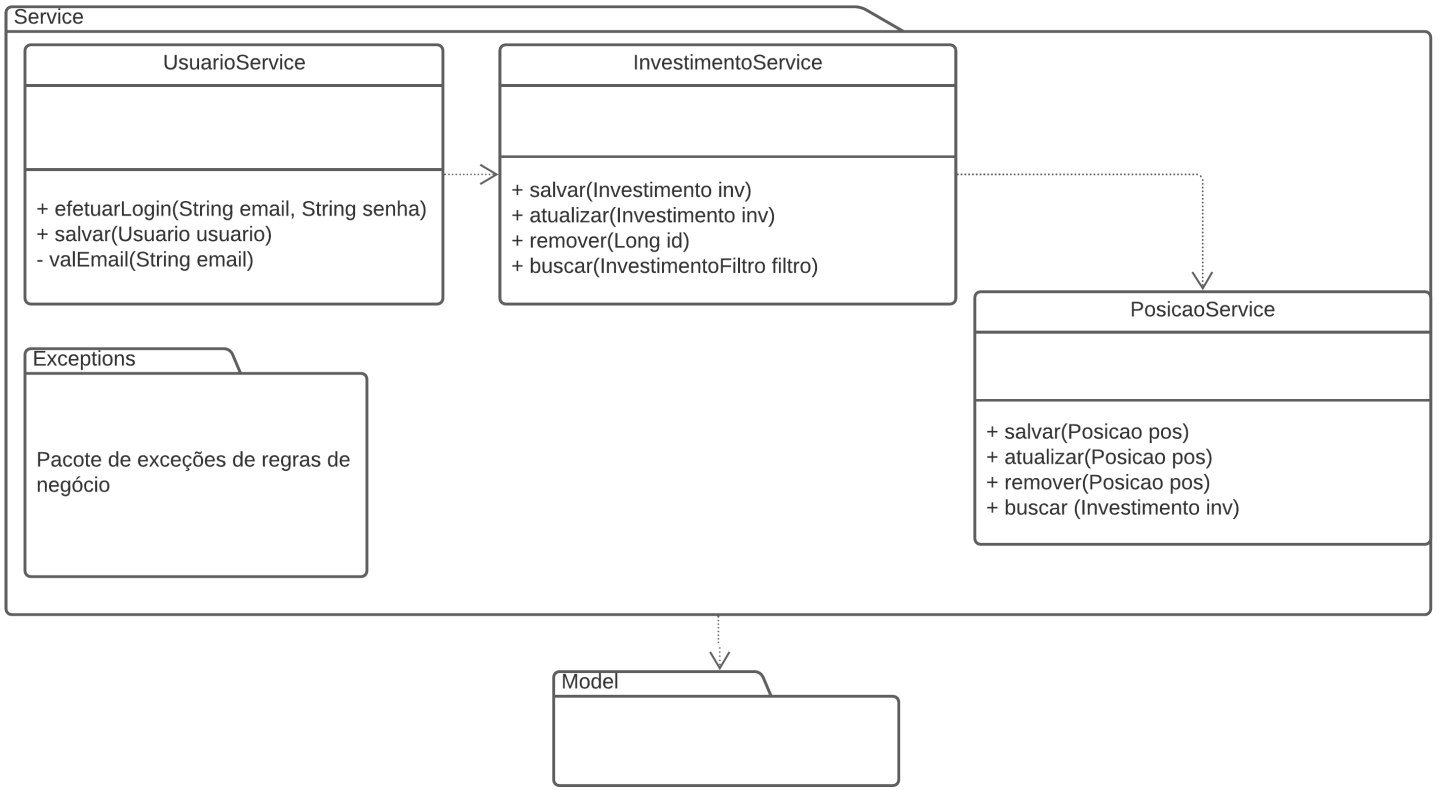
Visão de Casos de Uso



Visão Arquitetural



Visão de Classes



Implementando UsuarioService

Comportamentos esperados:

- efetuarLogin
  - informando email e senha válidos
- salvar
  - tratar se nome, email e senha são válidos
- obterSaldos
  - dos investimentos do usuário
  - para um usuário válido

Primeiro, criando a classe:

```
package com.labprog.patrimonio.service;

public class UsuarioService {

}
```

Para que seja um `Service` no SpringBoot e possamos utilizar o framework, inserimos uma anotação `@Service`

```
package com.labprog.patrimonio.service;

@Service
public class UsuarioService {

}
```

Agora adicionamos o repositório

- Lembrar de colocar a injeção de dependência com `@Autowired`

```
@Service
public class UsuarioService {

    @Autowired
    UsuarioRepository repository;
}
```

E vamos às assinaturas dos métodos

> “
>
> Para projetos maiores: uma opção é colocar esses métodos em `interfaces` visto que devem seguir o diagrama
>
> ”

```
@Service
public class UsuarioService {

    @Autowired
    UsuarioRepository repository;

    public boolean efetuarLogin(String email, String senha) {}
    public Usuario salvar(Usuario usuario) {}
    public List<InvestimentoSaldo> obterSaldos(Usuario usuario) {}
    private void verficarId(Usuario usuario) {}
    private void verificarUsuario(Usuario usuario) {}
}
```

Por partes: efetuarLogin

- Use o repositório para encontrar por email o usuário

```
public boolean efetuarLogin(String email, String senha) {
    Optional<Usuario> usr = repository.findByEmail(email);
```

- Adicionar em `UsuarioRepository` a função `findByEmail`

```
public interface UsuarioRepository extends JpaRepository<Usuari
    Optional<Usuario> findByEmail(String email);
}
```

- Observações:
  - o método retorna Optional: para não ter que tratar NullPointer
  - não é necessário implementar o método
  - esse método deve ser testado!

Por partes: efetuarLogin

- Com o usuário:

```
public boolean efetuarLogin(String email, String senha) {
    Optional<Usuario> usr = repository.findByEmail(email);
    if (!usr.isPresent())
        throw new RegraNegocioRunTime("Erro de autenticação. Em
    if (!usr.get().getSenha().equals(senha))
        throw new RegraNegocioRunTime("Erro de autenticação. Se

    return true;
}
```

Por partes: efetuarLogin

- Ok, adicionamos uma `RuntimeException` para padronizar o tratamento de `erros de Negócio`

```
package com.labprog.patrimonio.service.exceptions;

public class RegraNegocioRunTime extends RuntimeException{

    public RegraNegocioRunTime(String msg) {
        super(msg);
    }
}
```

- Pontos de qualidade:
  - trata mensagens alinhadas com a documentação do projeto
  - permite estender logs
  - permite construir testes com mais facilidade

Por partes: salvar

- Observação importante:
  - antes de salvar, precisa garantir que os campos informados estejam ok
  - forma simples, escreva um método de validação:

```
private void verificarUsuario(Usuario usuario) {
        if (usuario == null)
            throw new RegraNegocioRunTime("Um usuário válido deve s
        if ((usuario.getNome() == null) || (usuario.getNome().equal
            throw new RegraNegocioRunTime("Nome do usuário deve ser
        if ((usuario.getEmail() == null) || (usuario.getEmail().equ
            throw new RegraNegocioRunTime("Email deve ser informadc
        boolean teste = repository.existsByEmail(usuario.getEmail()
        if (teste)
            throw new RegraNegocioRunTime("Email informado já exist
        if ((usuario.getSenha() == null) || (usuario.getSenha().equ
            throw new RegraNegocioRunTime("Usuário deve possui senh
    }
```

Por partes: salvar

- Salvar se tornou simples

```
public Usuario salvar(Usuario usuario) {
    verificarUsuario(usuario);
    return repository.save(usuario);
}
```

- Para melhorar, vamos executar essa tarefa como uma transação

```
@Transactional
public Usuario salvar(Usuario usuario) {
    verificarUsuario(usuario);
    return repository.save(usuario);
}
```

Continuando: obtendo saldos

- O que sabemos:
  - o usuário pode ter muitos investimentos
  - um investimento pode ter muitas posições
  - o saldo final de um investimento é o somatório de todas as posições
- Voltamos ao `UsuarioRepository` para resolver

Continuando: obtendo saldos

```
public interface UsuarioRepository
      extends JpaRepository<Usuario, Long>{
        boolean existsByEmail(String email);
        Optional<Usuario> findByEmail(String email);

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

- Query explicita usando `JPQL`: Java Persistence Query Language
  - por querer retornar um objeto não mapeado teve que inserir o `InvestimentoSaldo`
- `@Param(<nome>` especifica parâmetro dentro da consulta

Continuando: obtendo saldos

- InvestimentoSaldo é apenas um DTO: Data Transfer Object
  - neste caso, banco -> aplicação

```
package com.labprog.patrimonio.model.dto;

import com.labprog.patrimonio.model.entidades.Investimento;
import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class InvestimentoSaldo {
    public Investimento inv;
    public Double valor;
}
```

Continuando: obtendo saldos

- Agora sim, vamos ao serviço:

```
public List<InvestimentoSaldo> obterSaldos(Usuario usuario) {
verficarId(usuario);
return repository.obterSaldosInvestimentos(usuario);
}
```

- ou se desejar uma variação:

```
public List<InvestimentoSaldo> obterSaldos(Long idUsuario) {
Optional<Usuario> usuario = repository.findById(idUsuario);
if (usuario.isPresent())
    return repository.obterSaldosInvestimentos(usuario.get());
throw new RegraNegocioRunTime("Usuario inválido");
}
```

Continuando: obtendo saldos

- O `verificarId` é apenas um check de consistência

```
private void verficarId(Usuario usuario) {
  if ((usuario == null) || (usuario.getId() == null))
      throw new RegraNegocioRunTime("Usuario inválido");
}
```

WoW: UsuarioSerice completo!

```
@Service
public class UsuarioService {

    @Autowired
    UsuarioRepository repository;

    public boolean efetuarLogin(String email, String senha) {
        Optional<Usuario> usr = repository.findByEmail(email);
        if (!usr.isPresent())
            throw new RegraNegocioRunTime("Erro de autenticação. Em
        if (!usr.get().getSenha().equals(senha))
            throw new RegraNegocioRunTime("Erro de autenticação. Se

        return true;
    }
```

WoW: UsuarioSerice completo!

```
@Transactional
public Usuario salvar(Usuario usuario) {
    verificarUsuario(usuario);
    return repository.save(usuario);
}

public List<InvestimentoSaldo> obterSaldos(Usuario usuario) {
    verficarId(usuario);
    return repository.obterSaldosInvestimentos(usuario);
}

public List<InvestimentoSaldo> obterSaldos(Long idUsuario) {
    Optional<Usuario> usuario = repository.findById(idUsuario);
    if (usuario.isPresent())
        return repository.obterSaldosInvestimentos(usuario.get(
    throw new RegraNegocioRunTime("Usuario inválido");
}
```

WoW: UsuarioSerice completo!

```
private void verficarId(Usuario usuario) {
        if ((usuario == null) || (usuario.getId() == null))
            throw new RegraNegocioRunTime("Usuario inválido");
    }

    private void verificarUsuario(Usuario usuario) {
        if (usuario == null)
            throw new RegraNegocioRunTime("Um usuário válido deve s

        if ((usuario.getNome() == null) || (usuario.getNome().equal
            throw new RegraNegocioRunTime("Nome do usuário deve ser

        if ((usuario.getEmail() == null) || (usuario.getEmail().equ
            throw new RegraNegocioRunTime("Email deve ser informadc

        boolean teste = repository.existsByEmail(usuario.getEmail()
        if (teste)
            throw new RegraNegocioRunTime("Email informado já exist

        if ((usuario.getSenha() == null) || (usuario.getSenha().equ
            throw new RegraNegocioRunTime("Usuário deve possui senh
    }
```

Investimento e Posição?

- Basta seguir o exemplo de UsuarioService
- Cria as assinaturas
- Define as restrições
- Implementa os metódos e às vezes, as funções nos repositórios

InvestimentoService: CRUD

```
@Service
public class InvestimentoService {

    @Autowired
    InvestimentoRepository repository;

    @Autowired
    UsuarioRepository usuarioRep;

    @Autowired
    PosicaoRepository posicaoRep;

    public Investimento salvar(Investimento inv){}
    public Investimento atualizar(Investimento inv) {}
    public void remover(Investimento inv) {}
    public List<Investimento> buscar (Investimento filtro) {}
    public Double obterValorTotal(Investimento inv){}
    private void verficarId(Investimento inv){}
    private void verificaInvestimento(Investimento inv) {}
}
```

InvestimentoService: checks

```
    private void verificarPosicao(Investimento inv) {
        List<Posicao> res = posicaoRep.findByInvestimento(inv);
        if (!res.isEmpty())
            throw new RegraNegocioRunTime("Investimento informado p
    }

    private void verificarId(Investimento inv) {
        if ((inv == null) || (inv.getId() == null))
            throw new RegraNegocioRunTime("Investimento sem id");
    }

    private void verificaInvestimento(Investimento inv) {
        if(inv == null)
            throw new RegraNegocioRunTime("Um investimento válido d

        if ((inv.getNome() == null)
            || (inv.getNome().equals("")))
        throw new RegraNegocioRunTime("Nome do investimento precisa

        if(inv.getUsuario() == null)
            throw new RegraNegocioRunTime("Um investimento deve est
```

InvestimentoService: checks

- Precisou adicionar `check` de usuário no repositório

```
public interface InvestimentoRepository
    extends JpaRepository<Investimento, Long> {

    public List<Investimento> findByUsuario(Usuario usuario);
}
```

InvestimentoService: salvar e atualizar

```
    public Investimento salvar(Investimento inv) {
        verificaInvestimento(inv);
        return repository.save(inv);
    }

    public Investimento atualizar(Investimento inv) {
        verficarId(inv);
        return salvar(inv);
    }
```

InvestimentoService: remover

```
    public void remover(Investimento inv) {
        verificarId(inv);
        verificarPosicao(inv);
        repository.delete(inv);
    }
```

- Alternativa só com ID

```
    public void remover(Long idInvestimento) {
        Optional<Investimento> inv = repository.findById(idInvestim
        remover(inv.get());
    }
```

InvestimentoService: buscar

- Vamos usar a classe `Example` do spring
  - preenche-se um objeto com os campos que deseja (nesse caso, nome do investimento)
  - deseja-se buscar por qualquer subnome (like)
  - ou por qualquer usuário

```
    public List<Investimento> buscar (Investimento filtro) {
        Example<Investimento> example =
          Example.of(filtro, ExampleMatcher.matching()
                .withIgnoreCase()
                .withStringMatcher(StringMatcher.CONTAINING)); //lik

        return repository.findAll(example); //o rep já deu essa imp
    }
```

InvestimentoService: obterValorTotal

- Precisa optar:
  - ou pega todas as posições e soma no código
  - ou contrói função no repositório

```
public interface InvestimentoRepository
    extends JpaRepository<Investimento, Long> {

    public List<Investimento> findByUsuario(Usuario usuario);

    @Query("select sum(p.valor) " +
        "from Posicao p join p.investimento i "+
        "where p.investimento = :investimento ")
    Double obterSaldoInvestimento(
            @Param("investimento") Investimento inv);
```

- Aqui foi bem mais simples do que em UsuarioRepository
  - bastou retornar um Double com o somatório

InvestimentoService: obterValorTotal

- Apenas validar e chamar o repositório

```
public Double obterValorTotal(Investimento inv) {
    verficarId(inv);
    return repository.obterSaldoInvestimento(inv);
}
```

PosicaoService: CRUD

- Muito parecido

```
@Service
public class PosicaoService {

    @Autowired
    PosicaoRepository repository;

    @Autowired
    InvestimentoRepository investimentoRep;

    public Posicao salvar(Posicao pos) {}
    public Posicao atualizar(Posicao pos) {}
    public void remover(Posicao pos) {}
    public List<Posicao> buscar (Posicao filtro) {}
}
```

PosicaoService: buscar

- Diferenças em buscar:
  - só tem o campo investimento
  - e data

```
public List<Posicao> buscar (Posicao filtro) {
Example<Posicao> example =
  Example.of(filtro, ExampleMatcher.matchingAny());

return repository.findAll(example);
}
```

Serviços implementados

Prox. aula: teste dos serviços