PROGRAMAÇÃO
DEFENSIVA

CONGRESSO DE TECNOLOGIA - FATEC

@mayulabs

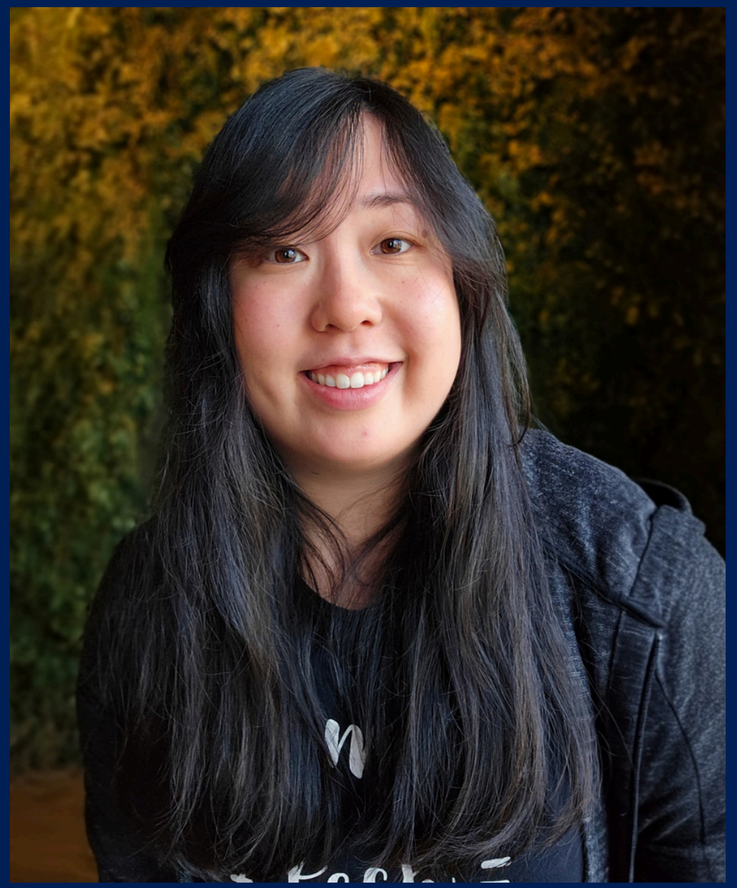
Eu sou a Mayu •

Deixa eu me apresentar pra você

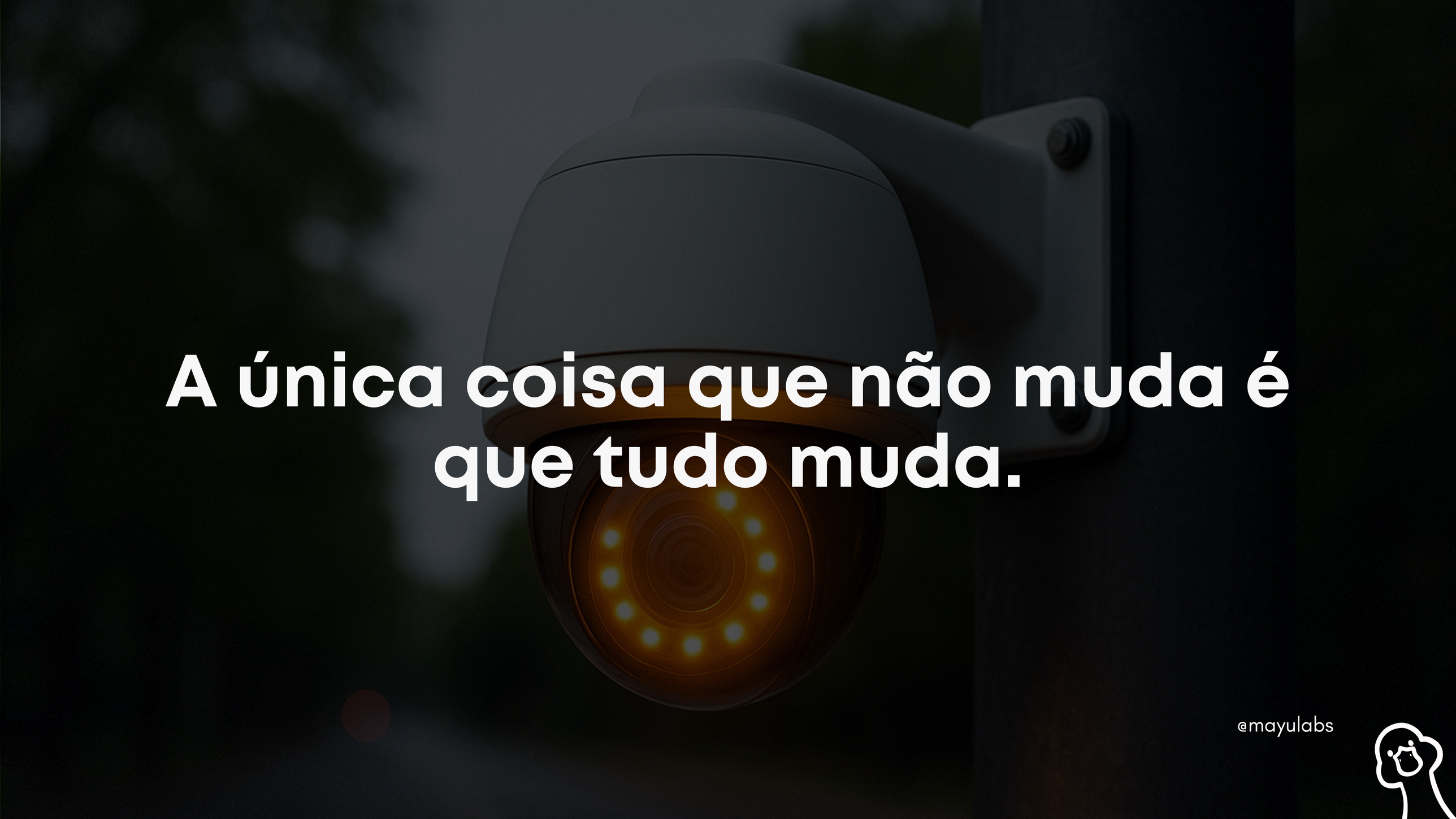
Comecei na tecnologia, por acidente, você acredita? Há 8 anos eu dei meus primeiros passos na computação e descobri minha paixão em programar.

E desde então?

Atualmente atuo como Engenheira de software Senior, Tech Lead no Itaú e trabalho com modernização de arquiteturas e sistemas há 5 anos. Aspirante a influenciadora, crio conteúdo sobre tecnologia no Instagram e Linkedin :)



@mayulabs


A única coisa que não muda é
que tudo muda.
@mayulabs

O que é Programação Defensiva?

Definição: Conjunto de técnicas para garantir a estabilidade e segurança do software em situações não previstas.

Objetivo: Diminuir falhas, evitar erros, garantir robustez e melhorar a experiência do usuário.

Prevenção: Reduzir bugs e vulnerabilidades desde o início da

Qualidade: Melhorar a manutenibilidade e confiabilidade do código.

Colaboração: Facilitar o trabalho em equipe e a compreensão do código.

Crescimento: Desenvolver uma mentalidade proativa na resolução de problemas.

Por que é importante?

Lição 1: Validação e Tratamento de Erros

Validação de Entradas

Sempre valide dados de entrada (usuário, APIs, arquivos)

Verifique tipo, formato, intervalo e segurança

```
// Exemplo de validação
if (!isValidEmail(email)) {
return "Email inválido";
}
```

Tratamento de Erros

Capture e trate exceções de forma controlada

Registre logs para depuração

```
try {
processarDados(dados);
} catch (erro) {
logger.error(erro);
return valorPadrao;
```

Lição 2: Assertions e Fail-Fast

Assertions

Validar suposições internas do código que *deveriam* ser verdadeiras.

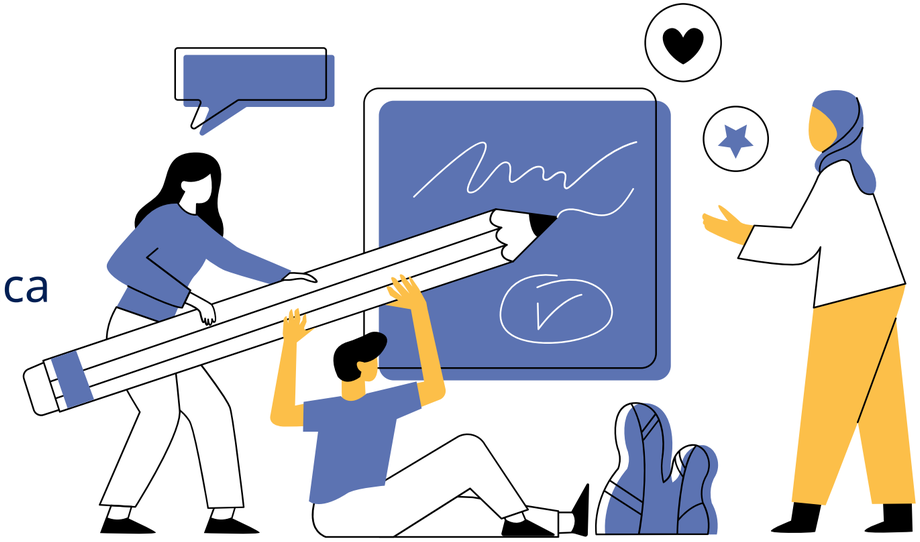
```
// Exemplo em JavaScript
assert(idade > 0, "Idade deve ser positiva");
assert(usuarios.length > 0, "Lista vazia");
```

⚠ Não use para validação de entrada de usuário em produção!

- É usado para verificar suposições internas do código que nunca deveriam falhar se o programa estiver correto.
- É uma ferramenta de debugging e não deve ser usada para validar entradas externas ou condições que podem variar em produção.
- Em algumas linguagens pode ser desativado, dependendo do modo como é executado, o que significa que ele não impactará o desempenho.

Benefícios:

- Detecta bugs durante o desenvolvimento
- Documenta suposições do código
- Facilita a depuração
- Melhora a qualidade do software



Lição 2: Assertions e Fail-Fast

Princípio Fail-Fast

Falhar o mais cedo possível quando uma condição inválida é detectada.

Facilidade de depuração:

- Detectar erros cedo ajuda a identificar a causa raiz mais rapidamente.

Redução de custos:

- Evita que erros pequenos se tornem problemas maiores e mais caros de corrigir.

Código mais confiável:

- O sistema se torna mais robusto, pois evita estados inválidos.

Sem Fail Fast:

```python
def processar_dados(dados):
    # Suponha que o programa continue mesmo com dados inválidos
    resultado = []
    for dado in dados:
        if isinstance(dado, int):
            resultado.append(dado * 2)
    return resultado

# Aqui, o programa ignora os dados inválidos, mas não informa o problema
print(processar_dados([1, "a", 3]))  # Saída: [2, 6]
```

Com Fail Fast:

```python
def processar_dados(dados):
    # Valida os dados no início e falha imediatamente se houver problema
    if not all(isinstance(dado, int) for dado in dados):
        raise ValueError("Todos os itens devem ser inteiros")
    return [dado * 2 for dado in dados]

# Aqui, o programa falha imediatamente ao detectar dados inválidos
print(processar_dados([1, "a", 3]))  # Lança ValueError
```

Lição 3: Não esqueça do básico

Código Limpo e Legível

Nomes descritivos, organização lógica e formatação consistente.

Princípio DRY (Don't Repeat Yourself)

Evitar duplicação de código através de funções e abstrações.

Comentários e Documentação

Explicar o porquê do código, não apenas o que ele faz.

Testes Unitários abrangentes

Garantir que o código funciona como esperado e que as defesas estão ativas.

```
// Exemplo ruim
function p(d) {
 return d * 2;
}

// Exemplo bom
function duplicarValor(numero) {
 return numero * 2;
}
```

Lição 4: Versionamento de API: Por que e Como?

O que é? Gerenciar e comunicar mudanças em APIs

Por que é importante?

- Clientes que utilizam versões antigas da API continuem funcionando sem problemas.
- Novas funcionalidades ou alterações possam ser introduzidas sem quebrar a experiência dos usuários existentes.

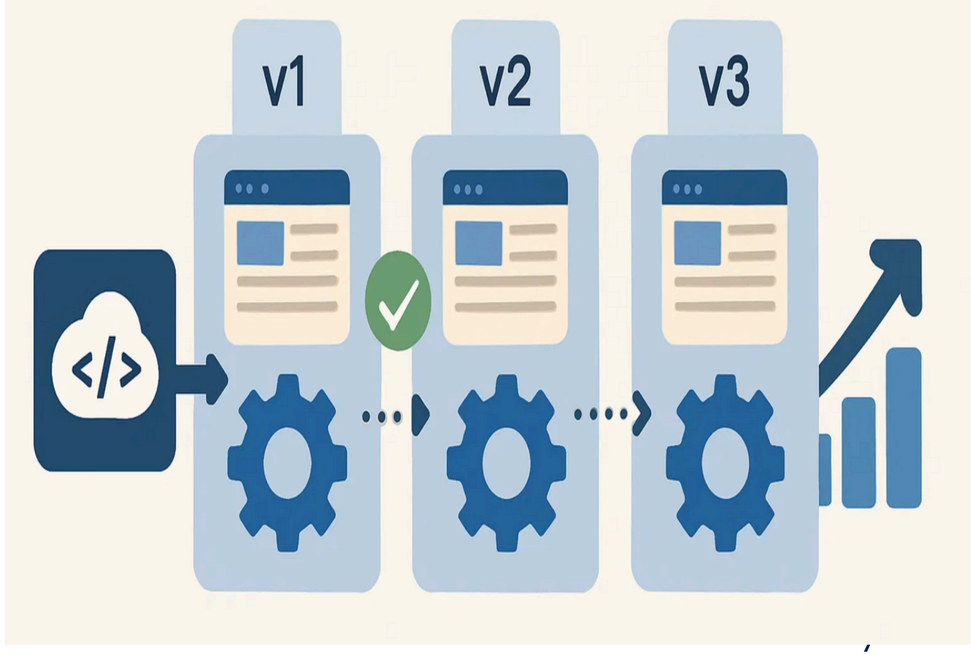
Estratégias Comuns:

URI Path: `/v1/users` (mais comum)

Custom Header: `X-API-Version: 1`

Accept Header: `Accept: application/vnd.myapi.v1+json`

Melhores Práticas: Documentar, comunicar e usar versionamento semântico (Major.Minor.Patch).



Lição 5: Termos pra guardar contigo

Feature flags:

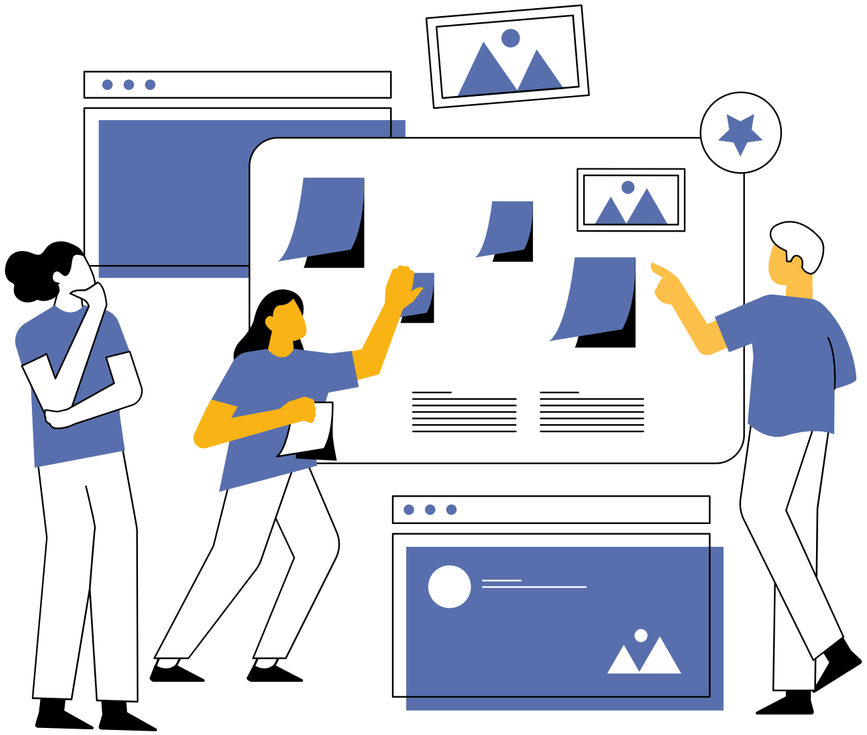
Ative ou desative funcionalidades em tempo de execução sem deploy.

Timeout e retry:

Configure limites de tempo e re-tentativas para chamadas externas.

Fallback

Forneça alternativas quando serviços externos falharem.



Lição 5: Termos pra guardar contigo

Deploy Gradual

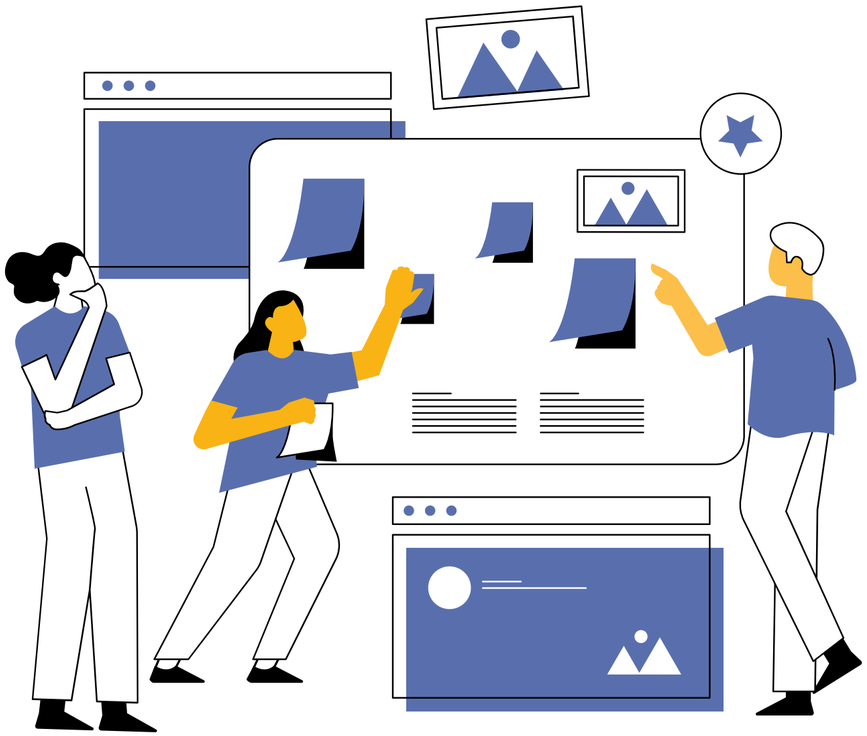
Libere novas versões para pequenos grupos antes de expandir. Escolha a que combina com sua app.

Simulação de Falhas:

Teste o comportamento da aplicação em cenários de falha.

Rollback Automático:

Reverta automaticamente versões problemáticas em produção.



Não esqueça tbm :)

Logs Estruturados

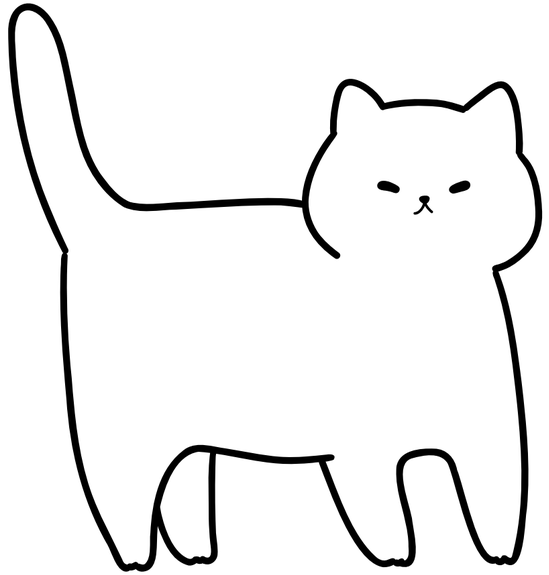
Registre logs detalhados para facilitar a depuração.

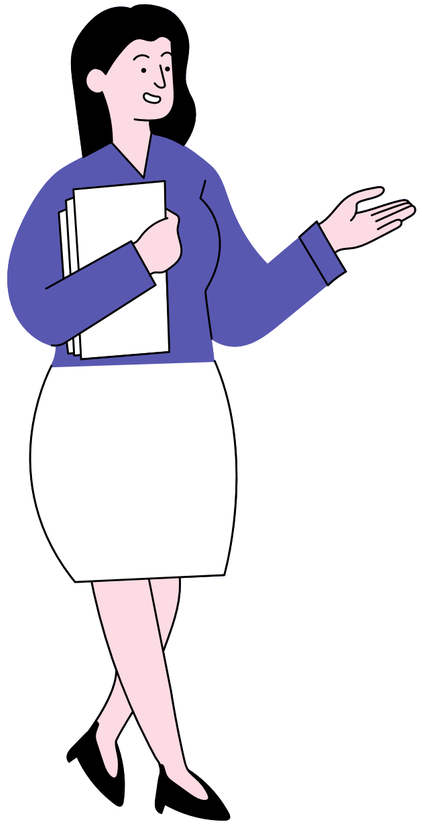
Testes Automatizados

Escreva testes para evitar regressões e validar mudanças.

Monitoramento e Alertas

Use ferramentas para rastrear métricas e notificar problemas.





Me adicione :)

THANK YOU

Até a próxima ♥