Perguntas Técnicas

1 – Esse é um código simples em “Portugol” com alguns erros. Sua missão é encontrar e corrigir os erros. Explique as correções feitas e porque se aplicam/foram escolhidas.

```
algoritmo exercício_1
var
   numero1, numero2: inteiro

inicio
   escreva("Digite o primeiro número: ")
   leia(numero1)

   escreva("Digite o segundo número: ")
   leia(numero2)

   soma = numero1 + numero2

   se soma > 10 então
      escreva("A soma é maior que 10!")

   se soma <= 10 então
      escreva("A soma é menor ou igual a 10!")

   escreva("O resultado da soma é: " + soma)
fim_algoritmo
```

Resposta

```
algoritmo "exercicio_1"

Var
   // Adição da declaracao da variavel soma
   numero1, numero2, soma: inteiro

Inicio
   escreva("Digite o primeiro número: ")
   leia(numero1)

   escreva("Digite o segundo número: ")
   leia(numero2)
   //Substituicao do operador = por <-
   soma <- numero1 + numero2

   se soma > 10 entao
     //Substitui escreva por escrevaL para quebra de linha.
     escrevaL("A soma é maior que 10")
   senao
     escrevaL("A soma é menor ou igual a 10!")
   //Inclusao do comando fimse para encerrar o bloco se-senão.
   fimse

   // Mudei a concatenação da string para incluir
```

```
   //diretamente a variável soma utilizando a vírgula.
   escreva("O resultado da soma é: ", soma)

//Mudei a palavra-chave fim_algoritmo para Fimalgoritmo.
Fimalgoritmo
```

Comentários:

Alteração 1: Adição da declaração da variável soma e definir como do tipo inteiro. Fiz essa alteração para armazenar o resultado da soma dos números, pois a variável soma não estava sendo declarada.

Alteração 2: Substituição do operador = por <- para atribuir o valor da soma das variáveis numero1 e numero2 à variável soma. O operador igual (=) não é usado para atribuição de valor no portugol.

Alteração 3: A chamada da função escreva foi alterada para escrevaL. Isso foi feito porque a função escrevaL é utilizada para escrever uma linha de texto com quebra de linha ao final. Assim, cada mensagem será exibida em uma linha separada.

Alteração 4: Foi incluído o comando fimse para encerrar o bloco condicional se-senão. O comando foi incluido para indicar o fim da estrutura condicional e garantir a correta execução do algoritmo.

Alteração 5: Mudei a concatenação da string foi modificada para incluir diretamente a variável soma utilizando a vírgula. Isso permite que o valor da soma seja exibido como parte da mensagem sem a necessidade de usar o operador de concatenação (+).

Alteração 6: A palavra-chave fim_algoritmo foi alterada para Fimalgoritmo. A alteração foi feita, para seguir a convenção da linguagem portugol, que utiliza a palavra "fim" seguida do nome do elemento (neste caso, "algoritmo") para indicar o fim de uma seção de código.

2 – Esse é um código funcional de uma integração entre duas ferramentas (tool1 e tool2) em Python. Esse código possui melhorias técnicas e não-técnicas que podem ser implementadas. Descreva as melhorias que você identificar pertinente e porque elas se aplicam/foram escolhidas.

```
import requests
from flask import Flask, jsonify, request
app = Flask(__name__)

@app.route('/integration', methods=['POST'])
def integrate_tools():
 try:
 # Recebe os dados da requisição
 data = request.get_json()

 # Verifica se as informações necessárias estão presentes
```

if 'tool1_data' not in data or 'tool2_data' not in data:
return jsonify({'error': 'Dados ausentes'}), 400

tool1_data = data['tool1_data']
tool2_data = data['tool2_data']

response = requests.post('https://api.tool1.com/integration', json=tool1_data)
if response.status_code != 200:
return jsonify({'error': 'Erro na integração com a Ferrramenta 1'}), 500

response = requests.post('https://api.tool2.com/integration', json=tool2_data)
if response.status_code != 200:
return jsonify({'error': 'Erro na integração com a Ferramenta 2'}), 500

Retorna a resposta da integração
return jsonify({'success': 'Integração realizada com sucesso'}), 200

except Exception as e:
return jsonify({'error': 'Erro interno do servidor'}), 500

if __name__ == '__main__':
app.run(host='0.0.0.0', port=5000)

Resposta

app.py

```
import os
import logging
import requests
from flask import Flask, jsonify, request
from dotenv import load_dotenv

# Carregamento das variáveis de ambiente
load_dotenv()

URL_API_1 = os.getenv('TOOL1_URL','http://localhost:5001/api1')
URL_API_2 = os.getenv('TOOL2_URL', 'http://localhost:5001/api2')

app = Flask(__name__)

# Configuração do logger
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Classe para lidar com erros de requisição
class RequestError(Exception):
    pass

# Classe para lidar com erros internos do servidor
class InternalServerError(Exception):
```

```
    pass

@app.route('/integration', methods=['POST'])
def integrate_tools():
    try:
        # Recebe os dados da requisição
        data = request.get_json()

        # Verifica se as informações necessárias estão presentes
        if 'tool1_data' not in data or 'tool2_data' not in data:
            raise RequestError('Dados ausentes')

        tool1_data = data['tool1_data']
        tool2_data = data['tool2_data']

        # Integração com a Ferramenta 1
        response = requests.post(URL_API_1, json=tool1_data)
        if response.status_code != requests.codes.ok:
            error_msg = f"Erro na integração com a Ferramenta 1:
{response.status_code} - {response.text}"
            logger.error(error_msg)
            raise RequestError(error_msg)

        # Integração com a Ferramenta 2
        response = requests.post(URL_API_2, json=tool2_data)
        if response.status_code != requests.codes.ok:
            error_msg = f"Erro na integração com a Ferramenta 2:
{response.status_code} - {response.text}"
            logger.error(error_msg)
            raise RequestError(error_msg)

        # Retorna a resposta da integração
        return jsonify({'success': 'Integração realizada com
sucesso'}), requests.codes.ok

    except RequestError as e:
        error_msg = f"Erro na solicitação: {str(e)}"
        logger.error(error_msg)
        return jsonify({'error': error_msg}),
requests.codes.bad_request

    except Exception as e:
        error_msg = f"Erro interno do servidor: {str(e)}"
        logger.error(error_msg)
        return jsonify({'error': error_msg}),
requests.codes.server_error


if __name__ == '__main__':
    app.run(host='127.0.0.1', debug=True, threaded=True)
```

Comentários

Alteração 1. Foram usadas variáveis de ambiente para maior flexibilidade, segurança, portabilidade, facilidade de manutenção e reutilização de código. Em comparação com a abordagem de definir valores diretamente no código, isso torna o sistema mais configurável, seguro e adaptável a diferentes ambientes e requisitos.

Alteração 2. Configuração de loggers para registrar mensagens de erro, permitindo um melhor acompanhamento de eventuais problemas.

Alteração 3. Manipulação de erros personalizados ao definir duas classes de exceção personalizadas, *RequestError* e *InternalServerError*, para lidar com erros específicos relacionados a requisições e erros internos do servidor.

Alteração 4. Verificação de dados, de modo que, antes de realizar as integrações com as ferramentas, o código verifica se as informações necessárias estão presentes nos dados recebidos na requisição. Caso contrário, é lançada uma exceção *RequestError* informando que os dados estão ausentes.

Alteração 5. Retorno de respostas HTTP adequadas, dessa forma, o código retorna respostas HTTP com códigos e mensagens apropriadas de acordo com o tipo de erro ocorrido.

Arquivos adicionados para melhoria da qualidade da API e Testes

.env

```
TOOL1_URL=http://localhost:5001/api1
TOOL2_URL=http://localhost:5001/api2
```

Api de Teste

apis_teste.py

```
from flask import Flask, jsonify, request

app = Flask(__name__)

@app.route('/api1', methods=['POST'])
def api1_endpoint():
    try:
        # Processa a lógica da api1
        data = request.get_json()

        return jsonify({'message': 'API 1 executada com sucesso'}),
200
    except Exception as e:
        return jsonify({'error': str(e)}), 500

@app.route('/api2', methods=['POST'])
def api2_endpoint():
    try:
        # Processa a lógica da API 2
        data = request.get_json()

        return jsonify({'message': 'API 2 executada com sucesso'}),
200
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001)
```

Projetei essa API para ser usada em conjunto com o código anteriormente fornecido. Funciona para testar a integração com as Ferramentas 1 e 2, utilizando as URLs definidas nas variáveis de ambiente TOOL1_URL e TOOL2_URL. Essas URLs apontam para os respectivos endpoints /api1 e /api2 nesta API.

Então, a função da API é receber solicitações nas rotas /api1 e /api2, processar a lógica específica de cada API e retornar uma resposta adequada. Assim, a API de teste atua como uma ponte entre as Ferramentas 1 e 2 e o sistema maior, permitindo a integração entre elas.

exec.sh

```
#!/bin/bash

env_name=$1
port=$2

if [ -e "$env_name" ]; then
      rm -rf $env_name
fi

python3 -m venv $env_name
echo "Subindo o ambiente virtual"
echo "Ativando o ambiente virutal 'alelo-venv'"
source $env_name/bin/activate
echo "Instalando Bibliotecas e Dependencias"
pip install -r requirements.txt > /dev/null
echo "Bibliotecas e Dependecias instaladas com Suceso!!"
echo "Executando a API de integração na porta $port"
python3 app.py -p $port
```

Fiz um shell script que possui a funcionalidade de configurar e executar as APIs construídas em conjunto com o ambiente virtual correspondente. Ele automatiza algumas tarefas necessárias para garantir que o ambiente esteja configurado corretamente e as dependências estejam instaladas.

requirements.txt

```
blinker==1.6.2
certifi==2023.5.7
charset-normalizer==3.1.0
click==8.1.3
Flask==2.3.2
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.3
```

```
python-dotenv==1.0.0
requests==2.31.0
urllib3==2.0.3
Werkzeug==2.3.6
```

O arquivo requirements.txt foi usado para listar as dependências do projeto. Ele serviu como um mecanismo para especificar as bibliotecas e suas versões necessárias para que o projeto funcionasse corretamente.

Testes Unitários

test_api.py

```
from flask import *
import pytest
from app import app


@pytest.fixture
def client():

    with app.test_client() as client:
        yield client

def test_integration_endpoint(client):
    # Defina os dados para a integração
    data = {
        'tool1_data': {
            'key1': 'value1',
            'key2': 'value2'
        },
        'tool2_data': {
            'key3': 'value3',
            'key4': 'value4'
        }
    }

    # Faça uma requisição POST para a rota da função de integração
    response = client.post('/integration', json=data)

    # Verifique o código de status da resposta
    assert response.status_code == 200

    # Verifique o conteúdo da resposta
    response_data = response.get_json()
    assert 'success' in response_data
    assert response_data['success'] == 'Integração realizada com
sucesso'


def test_integration_endpoint_invalid_data(client):
    # Defina os dados para a integração (dados inválidos)
    data = {
        'tool1_data': {
            'key1': 'value1',
            'key2': 'value2'
        }
```

```
    }

    # Faça uma requisição POST para a rota da função de integração
    response = client.post('/integration', json=data)

    # Verifique o código de status da resposta
    assert response.status_code == 400, f"Código de status
inesperado: {response.status_code}"

    # Verifique o conteúdo da resposta
    response_data = response.get_json()
    assert 'error' in response_data, "A chave 'error' não está
presente na resposta"


def test_integration_endpoint_internal_error(client):

    def integration_function():
        raise Exception("Erro interno do servidor: 400 Bad Request:
Failed to decode JSON object: Expecting value: line 2 column 1 (char
1)")

    app.integration = integration_function

    response = client.post('/integration', json=None)
    response.headers.add('Content-Type', 'application/json')
    assert response.status_code == 500, f"Código de status
inesperado: {response.status_code}"

    response_data = response.get_json()
    assert 'error' in response_data, "A chave 'error' não está
presente na resposta"


def test_integration_endpoint_tool1_error(client):
    # Defina os dados para a integração
    data = {
        'tool1_data': {
            'key1': 'value1',
            'key2': 'value2'
        },
        'tool2_data': {
            'key3': 'value3',
            'key4': 'value4'
        }
    }

    # Faça uma requisição POST para uma rota inválida
    response = client.post('/invalid-route', json=data)

    # Verifique o código de status da resposta
    assert response.status_code == 404, f"Código de status
inesperado: {response.status_code}"


def test_integration_endpoint_tool2_error(client):
    # Defina os dados para a integração
    data = {
        'tool1_data': {
            'key1': 'value1',
            'key2': 'value2'
```

```
        },
        'tool2_data': {
            'key3': 'value3',
            'key4': 'value4'
        }
    }
    # Faça uma requisição POST para uma rota inválida
    response = client.post('/invalid-route', json=data)

    # Verifique o código de status da resposta
    assert response.status_code == 404, f"Código de status
inesperado: {response.status_code}"
```

Fiz os casos de teste para a API de integração. Ele usa o framework de testes pytest para definir e executar os casos de teste. Assim, posso garantir a qualidade e o funcionamento correto da API. Eles ajudam a identificar e corrigir erros, validar o comportamento esperado do sistema e fornecer confiança de que as funcionalidades estão implementadas corretamente.

Segue o link do github: Alelo-technical-challenge

3 – A empresa VendeMais é do ramo de e-commerce, possuem um site de compras e um sistema interno de gerenciamento de pedidos e estoque. A VendeMais deseja integrar seus sistemas com a EntregoFácil, um provedor de serviços de logística, para automatizar o processo de envio e rastreamento de pedidos.
Como você imagina que essa integração pode ser feita? Descreva em alto nível os passos para isso e desenhe um diagrama simples para ilustrar seu raciocínio.
OBS: Não se preocupe com perfeição, o objetivo é avaliar a sua noção lógica/sistêmica e fundações técnicas.

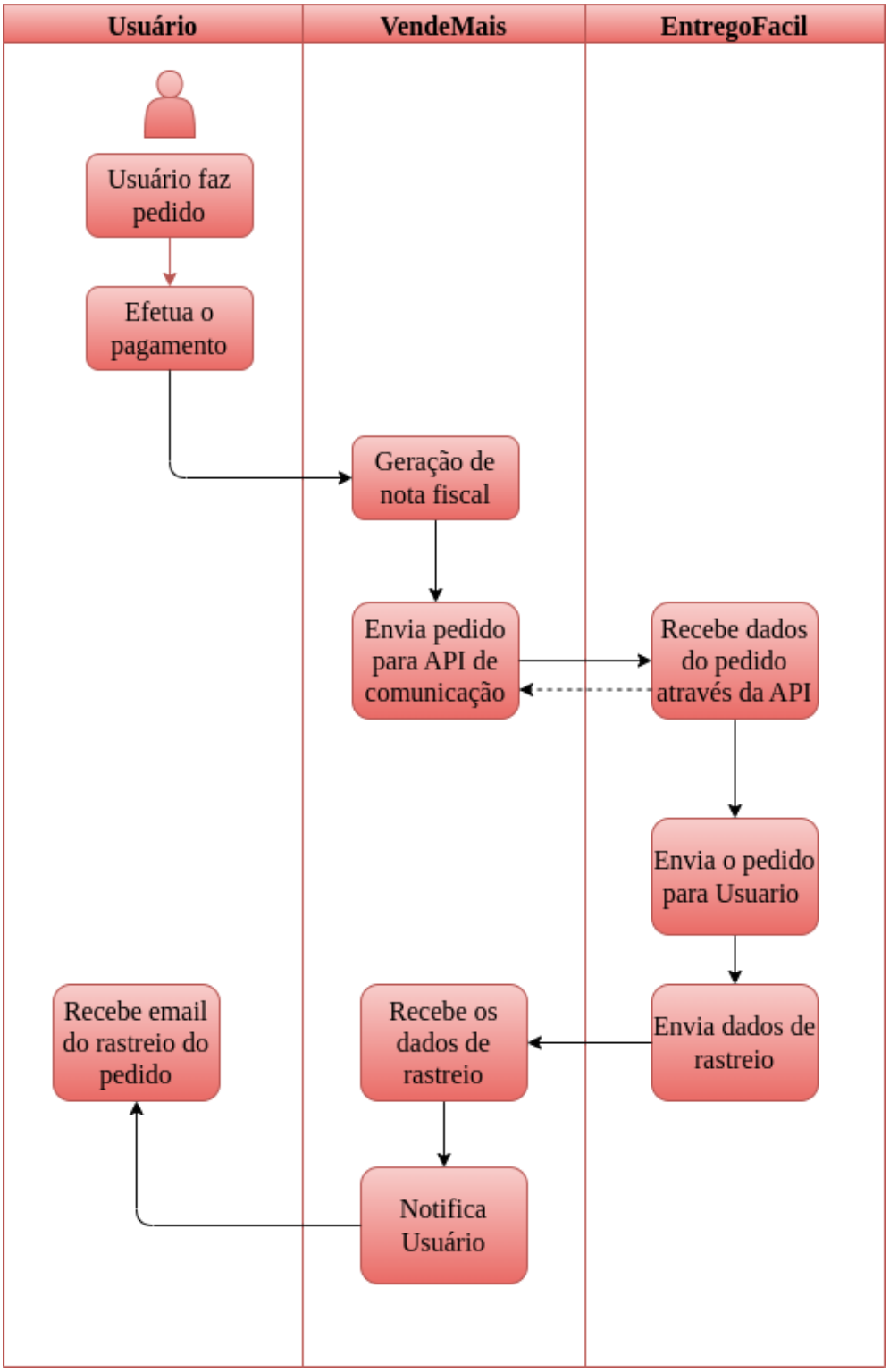
Inicialmente serão definidos os requisitos dessa integração, por meio de reuniões entre a VendeMais e a EntregoFacil. Assim, será possível identificar quais informações precisam ser trocadas entre os sistemas da VendeMais e da EntregoFácil. Isso inclui detalhes dos pedidos, informações de entrega, dados de rastreamento, entre outros.

Em seguida é preciso estabelecer uma comunicação entre os sistemas da VendeMais e da EntregoFácil, que será realizada por meio de uma API (Application Programming Interface). A API será responsável por permitir a troca de informações entre os sistemas. Além disso, implementar um mecanismo de segurança para garantir que apenas sistemas autorizados possam acessar e trocar informações pela API. E também definir como os dados serão estruturados e mapeados entre os sistemas ao estabelecer uma correspondência entre os campos de informação dos pedidos, endereços de entrega, produtos, status de entrega, etc., nos sistemas da VendeMais e da EntregoFácil.

Quando um usuario faz um pedido no site da VendeMais, o sistema interno de gerenciamento de pedidos gera a nota fiscal e deve enviar os detalhes do pedido para a EntregoFácil por meio da API que serão informações de endereço de entrega, produtos comprados e quantidade .

Então, a EntregoFácil recebe os detalhes do pedido e realiza o processamento de envio. Após isso, o pedido é enviado ao usuario e a medida que a EntregoFácil realiza o transporte e entrega do pedido, ela atualiza o status de entrega. Essas atualizações de status devem ser enviadas de volta para o sistema da VendeMais por meio da API. Assim, os clientes da VendeMais poderão acompanhar o status de entrega de seus pedidos.

Os clientes da VendeMais poderão acompanhar o status de entrega de seus pedidos por meio do site de compras e receberão notificações por e-mail. Isso é possível porque o sistema da VendeMais consulta a API da EntregoFácil para obter as informações atualizadas de rastreamento.

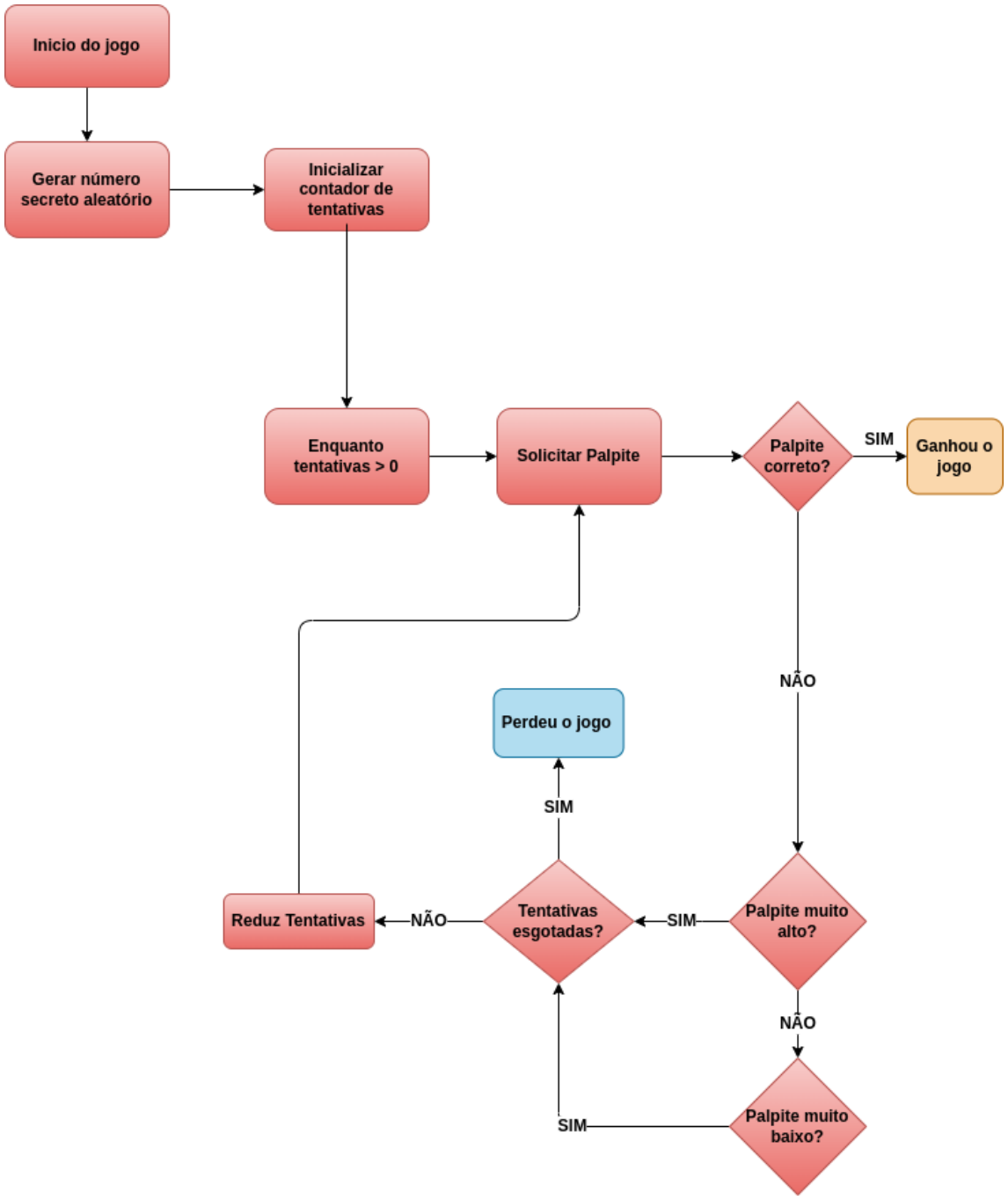


4- Desenvolver um fluxograma que implemente um jogo de adivinhação. O jogador terá 7 tentativas para adivinhar qual é esse número.

Requisitos:

- Gerar um número secreto aleatório entre 1 e 100.
- A cada tentativa, o jogador deve fornecer um palpite.

- Informar ao jogador se o palpite está correto, muito alto ou muito baixo em relação ao número secreto.
- O jogo deve ser encerrado quando o jogador acertar o número ou esgotar o número de tentativas.



Perguntas Pessoais

OBS: Não tem certo/errado, são apenas perguntas para te conhecermos melhor.

1 – Qual sua expectativa com a vaga? Como você se imagina atuando e o que gostaria de fazer?

Tenho altas expectativas, já faz um tempo que me interessei por Devops, desde o meu último trabalho, onde pude fazer uma automações de entrega contínua. Eu me imagino criando pipelines de automação para entrega de código, utilizando diversas tecnologias como terraform, github actions, airflow, jenkins e também gostaria de aplicar meus conhecimentos em cloud.

2 – O que você conhece sobre DevOps? Já fez algo/trabalhou em algo relacionado? (Mesmo que seja estudo, laboratórios, projetos pessoais, etc).

O Devops orquestra, adminstra, monitora e automatiza entregas continuas de software, conheço como a magia da infraestrutura. Sei também que o DevOps é uma abordagem cultural e prática que visa a colaboração e integração entre equipes de desenvolvimento (Dev) e operações (Ops) para melhorar a eficiência e a qualidade na entrega de software. Envolve a automação de processos, o uso de ferramentas e a adoção de práticas ágeis. Já trabalhei como desenvolvedora backend em uma startup e lá construi pipelines para entregas continuas.

3 – Como é sua rotina de estudos? O que você tem estudado? O que pretende estudar como próximos passos?

Atualmente minha rotina de estudos é estudar para certificação aws developer e também participo de um bootcamp de desenvolvimento full stack, onde faço entregas semanais de projeto. Pela manhã estudo para tirar a certificação e trabalho no projeto do bootcamp e a tarde continuo com a construção dele. À noite tenho aula do bootcamp. Como próximos passos, pretendo tirar a certificação da aws e seguir a trilha de Devops, bem como me especializar no Devops.

OBS: Todos os exercícios estão no meu github, no link a seguir: Alelo-technical-challenge.