Avaliação técnica e conceitual

Desenvolvedor Web FullStack

# Observações e orientações

* A será realizada em duas etapas:
* Um teste de conceitos gerais de programação, paradigmas de programação e conceitos básicos de SQL;
* Um teste prático de linguagens de programação.
* Não é necessário concluir toda a avaliação. Avaliaremos sua capacidade de interpretação e do desenvolvimento das respostas;
* Na avaliação prática será avaliado tanto o resultado quanto a sua lógica, organização e clareza de código;
* Você deve resolver as questões utilizando **Python**;
* **Boa prova!**

Nome Completo: Zaqueu Caetano Mariano dos Santos

Data Entrega : 26/05/2021

Tempo gasto para realização da prova Conceitual: 1:30 horas

Tempo gasto para realização da prova Prática: 12 horas

Etapa 1

### Avaliação conceitual

1. Qual a principal diferença entre os paradigmas de programação **procedural** e **orientada a objetos**? O que é **classe** e **instância**? Em quê um **método** e uma **função** se diferenciam? Por que a orientação a objetos é importante em grandes aplicações? Em outras palavras, qual a principal vantagem de se trabalhar com objetos? Dê alguns exemplos (pode ser em esquemas ou em pseudo-código).

RESPOSTA 1= Primeiro precisamos entender que em alguns casos os paradigmas vem de uma evolução e em outros casos de uma modificação de outros paradigmas. Com a evolução em programação os procedimentos e funções não eram mais o suficiente para solucionar problemas já conhecidos, era preciso reescrever boa parte deles em outras funções, foi quando o polimorfismo surgiu na programação com a orientação a objetos, como uma espécie de assinatura invocando métodos com a mesma identificação, onde duas ou mais classes são derivadas de uma mesma superclasse.

1. O que são **APIs**? Dê ao menos um exemplo de tipo de API. Você sabe dizer onde elas costumam aparecer?

RESPOSTA 2 = A API é um conjunto de rotinas e padrões de programação para acesso a softwares ou plataformas web. Várias são as utilizações das APIs, como por exemplo nos plug-ins desenvolvidos para complementar a funcionalidade de um programa. Os sistemas operacionais possuem suas APIs, tornando possível que um usuário execute algum programa que envolva determinado processo do sistema operacional. Outro exemplo são os sistemas de pagamentos online, os quais também usam as funcionalidades da API. Ressaltando, por fim, que ela funciona através da comunicação de diversos códigos, definindo comportamentos específicos de determinado objeto em uma interface.

1. Descreva a arquitetura **cliente-servidor** num desenvolvimento de uma aplicação. Qual a vantagem de utilizá-la frente a outras arquiteturas? Cite exemplos de serviços que a utilizam hoje em dia.

RESPOSTA 3 = O processamento da informação é dividido em módulos. Enquanto um processo é responsável pela manutenção da informação (servidor), o outro é responsável pela obtenção dos dados (cliente). Como exemplo temos a comunicação entre o browser e um site que possui um servidor para processamento de banco de dados.

1. Numa aplicação estruturada na arquitetura cliente-servidor que você acabou de descrever, é muito comum desenvolver o servidor no “padrão de desenho” (*design pattern*) ***Model-View-Controller*** (**MVC**) ou então ***Model-View-Presenter*** (**MVP**). Escolha um dos dois e disserte um pouco a respeito. Você conhece algum framework que utilize um dos dois?

RESPOSTA 4= Com a MVC é possível separar a interface do Usuário das regras de negócio, isso se torna uma grande vantagem pela reutilização de códigos mesmo que sejam projetos diferentes, sem contar a facilidade na manutenção das aplicações.

1. Dado um banco SQL qualquer, MySQL por exemplo, denominado **EXEMPLO\_DB** e uma tabela **EMPREGADO** com as colunas:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **NOME** | **IDADE** | **SALARIO** |
| 1 | José | 34 | 4324.24 |
| 2 | Maria | 25 | 3600.00 |
| 3 | Antônio | 19 | 2100.15 |

Responda:

1. Quem poderia ser uma chave primária?

**R: ID**

1. Como se faria uma consulta para retornar todos os registros?

**R: SELECT \* FROM EMPREGADO;**

1. Como se faria uma consulta para retornar apenas os campos NOME e SALARIO de quem tem IDADE < 30?

**R: SELECT NOME, SALARIO FROM EMPREGADO WHERE IDADE = 30;**

1. Como atualizar o salário de Maria para 4000.00?

**R: UPDATE EMPREGADO SET SALARIO = ‘4000.00’ WHERE ID = ‘2’;**

1. Antônio foi demitido. Como excluir seu registro do banco?

**R: DELETE FROM EMPREGADO WHERE ID = ‘3’;**

1. O que é um mapa objeto relacional? Supondo que temos uma tabela do exercício acima, dê um exemplo de como ele seria utilizado numa aplicação (pode ser algumas linhas de pseudo-código).

RESPOSTA 6 = É uma técnica cujo objetivo é otimizar a Programação Orientada a Objetos por meio dos bancos de dados relacionais. Não é necessária uma correspondência direta entre as tabelas de dados e as classes do programa.

1. Qual a principal diferença entre bancos relacionais (SQL) como MySQL e não-relacionais (NoSQL) como MongoDB? Como são chamadas as entidades de um banco não-relacional?

RESPOSTA 7 = Bancos de dados relacionais representam e armazenam dados em tabelas e filas. Já os bancos de dados não relacionais não precisam ser divididos em tabelas, podendo ser em uma estrutura única.

1. Python: O que são *list comprehensions*? O que são dicionários? Como iterar sob todos os itens de um dicionário, imprimindo chave-valor? Como declarar uma classe?

RESPOSTA 8 = O list comprehensions no Python permite a criação e processamento de listas de forma limpa e fácil. Em Python, dicionário é um tipo de estrutura de dados do tipo coleção, ou seja, um objeto que contém mais que um valor

1. Cite e descreva algumas das boas práticas de programação.
2. O que caracteriza programação assíncrona? Qual sua vantagem em códigos que dependem de muitas chamadas de IO? Cite alguns exemplos de casos em que essa forma é vantajosa.
3. Git: O que é **git**? Quais são alguns comandos básicos? Você conhece algum esquema de trabalho que utilize o git?

RESPOSTA 11 = Sistema de controle de versão distribuído, utilizado no dia a dia do desenvolvedor, principalmente em DevOps. Alguns dos principais comandos do git são:

O **Git clone** é um comando para baixar o código-fonte existente de um repositório. git clone <https://nome-do-link>

Pode-se usar o comando **git branch** para criar, listar e excluir *branches*. Criando uma nova *branch*: git branch <nome-do-branch>

Usamos o **git checkout** principalmente para alternar de um branch para outro. Também podemos usá-lo para verificar arquivos e commits: git checkout <nome-da-branch>

1. Frontend: Cite alguns exemplos de *frameworks* de javascript usuais para desenvolver camadas de frontend para o usuário.
2. Frontend: Bibliotecas muito utilizadas em *frameworks* frequentemente fazem uso de estruturas de dados assíncronas do que se é chamado de **extensões reativas**. Dentre elas, a biblioteca RxJS se destaca e traz a estrutura de *Observables*. O quê são eles? Como se diferem de promessas?
3. Explique com suas palavras o que são microsserviços e qual o funcionamento principal do docker. Liste algumas vantagens de se utilizar esse ambiente.
4. [BÔNUS] Frontend: Você conhece a biblioteca de estados reativos NgRX? Qual a vantagem de se trabalhar com estados numa aplicação de frontend?
5. [BÔNUS] No python e javascript, é muito usual trabalhar com **ambientes virtuais** (*virtual environments*). Descreva o que são, para que servem o por que é uma boa prática de programação.

Etapa 2

### Avaliação prática

Nessa etapa você deverá utilizar o **Python** para resolver as questões práticas referentes à programas e ao backend.

Na questão que envolve frontend, você está livre para escolher qualquer framework, contudo **bonificaremos a escolha de Angular**.

**Desenvolva suas soluções em repositórios git.**

#### Programação funcional

1. Calculadora simples.
   * Implemente uma solução capaz de realizar as seguintes operações: soma, subtração, divisão, multiplicação, módulo, exponenciação, média entre dois valores e fatorial, assegurando-se de tratar quaisquer possíveis erros na implementação de cada função;
   * Faça uma pequena interface de linha de comando (CLI) para execução dessas funções.

**BÔNUS:** Implemente a calculadora utilizando orientação a objetos.

#### Programação orientada a objetos

1. Você quer representar uma partida de futebol. Em especial, estamos interessados em programar uma rodada de pênaltis.

Sabemos que uma dada partida é composta dois times, cada time é composto por um goleiro, um técnico e 10 jogadores. Cada jogador tem uma ação **chutar** que lança a bola ao gol. Para simplificar, considere que um dado chute de ambos os times tem 66% de chance de entrar no gol.

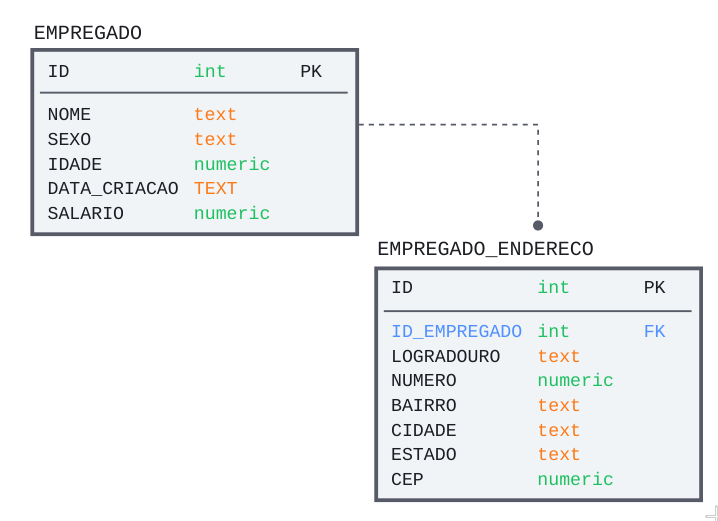
* + Crie classes para representar os jogadores e goleiro (lembre o que goleiro também é um jogador), técnico e do time;
  + Explique as relações de hierarquia;

**BÔNUS:** Implemente uma classe de jogo para rodar e armazenar as pontuações de uma partida de pênaltis.

#### Conexão com banco

1. A tabela do exercício 5 da primeira etapa está disponível num arquivo **SQLite** anexado a esse e-mail

Nela você encontrará as tabelas **EMPREGADO** e **EMPREGADO\_ENDERECO** contendo os mesmos registros da questão mencionada, porém com mais outras colunas, essas definidas:



Você é livre para **utilizar qualquer biblioteca** já pronta.

Escreva um programa que crie uma conexão com o banco e faça consultas, escrevendo o resultado na tela e que também consiga fazer inserções, atualizações e deletar registros. Utilize funções ou métodos para isso;

**BÔNUS:** Alternativamente faça a implementação utilizando mapas objeto-relacionais.

#### Elaboração de API

1. Utilizando alguma biblioteca de sua escolha (ou implementando na unha mesmo), faça uma API RESTful que implemente os seguintes endpoints, todos retornando JSON. Os exemplos de saída são uma sugestão, você está livre de implementar da forma que achar melhor. **Bônus** se você utilizar o Django.

Endpoints **hello** retornam uma mensagem:

* + GET /hello

-> {‘mensagem’: ‘Olá! Como vai?’}

* + GET /hello?nome=nome

-> {‘mensagem’: ‘Olá {{nome}}! Seja bem vindo!’}

Endpoints **lista** manipulam uma lista encadeada internamente:

* GET /lista

Retorna a lista como um todo

-> {‘lista’: [{{elementos da lista}}]}

* GET /lista/{{**elemento**}}

Checa se o **elemento** pertence a lista

-> {‘inlist’: true/false}

* POST /lista/{{**elemento**}}

Adiciona o **elemento** na lista

-> {‘sucesso’: true}

-> {‘sucesso’: false, ‘mensagem’: ‘{{**elemento**}} já na lista’}

* DELETE /lista/{{**elemento**}}

Remove o **elemento** da lista

-> {‘sucesso’: true}

-> {‘sucesso’: false, ‘mensagem’: ‘{{**elemento**}} não está na lista. Nada feito’}

1. Implemente uma API RESTful para manipulação da tabela da questão 3 (adicionar, listar, editar e remover usuários) com filtros e paginação na estrutura de algum banco de dados da sua escolha. Escreva também uma camada frontend para se comunicar com o servidor da API. **Bônus:** escreva também Dockerfiles e stacks de microsserviços para deploy rápido.