Teste

Importante:

- Ler com atenção as questões
- Não iremos avaliar uma linguagem específica
- Estamos abertos para perguntas, afinal nem toda especificação é clara e livre de ambiguidade

1) O código abaixo faz uma requisição ao site dos correios para obter o logradouro e bairro a partir de um cep:

```
def cep_json_xml(cep, formato):
    """ dado um <cep> retorna o endereço no <formato> especificado """
    cepEntrada = cep
    tipoCep    =""
    cepTemp    =""
    metodo     ="buscarCep"
    formato    = formato.lower()
    url = 'http://m.correios.com.br/movel/buscaCepConfirma.do'
    post_data_dictionary = {'cepEntrada': cepEntrada, 'tipoCep': tipoCep, 'cepTemp': cepTemp, 'metodo':metodo}
    #codifica os dados POST para ser enviado em uma URL
    post_data_encoded = urllib.urlencode(post_data_dictionary)
    try:
        #objeto request que armazena os dados do POST e da URL
        request_object = urllib2.Request(url, post_data_encoded)
        #faz o request usando o objeto request como um argumento e armazena a resposta em uma variavel
        response = urllib2.urlopen(request_object)
        #armazena a resposta em uma string
        result = response.read()
        #extrair dados de arquivos HTML (result)
        soup = BeautifulSoup(result)
        #seleciona as tags com essas classes
        values = soup.select(".caixacampobranco span.respostadestaque ")

        if len(values) > 2:
            resultado=1
            resultado_txt = "Sucesso cep completo"
            #remove outras informacoes que vem junto ao logradouro. Exemplo: - de 1000 a 2000 e impar
            logradouro = (values[0].text.strip()).split('-')
            logradouro = logradouro[0]
            #extrai os valores das tags
            bairro = values[1].text.strip()
            cidade_estado = values[2].text.split()
            cidade = cidade_estado[0]
            estado = cidade_estado[1].strip('/')
            cep = values[3].text.strip()
        elif len(values) < 2:
            resultado=2
            resultado_txt = "Sucesso cep unico"
            logradouro = ""
            bairro = ""
            cidade_estado = values[0].get_text().split()
            cidade = cidade_estado[0]
            estado = cidade_estado[1].strip('/')
            cep = values[1].get_text().strip()
        else:
            resultado_txt = "Servico indisponivel ou cep invalido"
    except:
        exc_type, exc_obj, exc_tb = sys.exc_info()
        file = exc_tb.tb_frame.f_code.co_filename
        retorno = json.dumps({'msg': 'type: %s, error: %s, file: %s, line %s' % (exc_type, exc_obj, file, exc_tb.tb_lineno)})
        return retorno,formato
    dict_json = {'resultado': resultado,
                 'resultado_txt': resultado_txt,
                 'logradouro' :logradouro,
                 'bairro': bairro,
                 'cidade': cidade,
                 'uf': estado,
                 'cep':cep}

    if formato == 'json':
        retorno = json.dumps(dict_json)
    elif formato == 'xml':
        retorno = open(os.path.join(os.path.dirname(__file__), 'cep.xml'),'r').read()%dict_json
    else:
        retorno = "Opcao nao existe"
    return retorno,formato
```

1.1) Do ponto de vista de qualidade de código, você identifica algum problema? Poderia listar alguns?

1.2) É possível ou é fácil escrever um teste para o código acima? Por quê?

1.3) Você poderia reescrever parcialmente ou todo o código acima afim de melhorar os problemas listados? (Queremos saber como você faz o refactoring e não testar se o código funciona)

2) Dado o método que calcula o fibonnaci de N:

```python
def fib(n):
    """
    Retorna fibonnaci de N
    """
    if n < 2:
        return 1
    return fib(n-1) + fib(n-2)
```

O código acima funciona perfeitamente, porém imagine a seguinte execução:

fib(125)

fib(126)

fib(125)

Na segunda execução (126), parece que poderíamos aproveitar a execução anterior (125) ao invés de reprocessar os números já calculados. Qual seria uma abordagem para evitar este desperdício de execuções?

3) Em um processamento distribuído, temos N workers que executam um trabalho e ao final fazem chamadas ao centralizador de execuções. A cada chamada é passado o identificador da execução, o item com seu estado e alguns parâmetros, exemplo:

JAN21-A, Item1, STARTED, {'params': ...}, None

JAN21-A, Item2, STARTED, {'params': ...}, None

JAN21-A, Item3, STARTED, {'params': ...}, None

JAN21-A, Item1, SUCCESS, {'params': ...}, {'result': 0.95}

JAN21-A, Item3, SUCCESS, {'params': ...}, {'result': 0.42}

JAN21-B, Item1, STARTED, {'params': ...}, None

JAN21-B, Item2, STARTED, {'params': ...}, None

...

Imagine que existe uma api http que recebe as chamadas dos workers, que por sua vez executa na camada de serviços o código abaixo:

```python
def update_execution(identifier, item_id, status, params, result):

    execution = db.session.query(Execution).filter(
        Execution.identifier == identifier
    ).one()

    created, execution_task = _get_or_create_execution_item(
        execution.id, item_id
    )

    execution_task.status = status
    if status == 'STARTED':
        execution.started_at = datetime.now(timezone.utc)

    if result and not execution.results:
        execution.results = [result]
    elif result and execution.results:
        execution.results.append(result)

    db.session.add(execution)
    db.session.add(execution_task)
    db.session.commit()
    return execution_task
```

Nota: Perceba que cada worker faz pelo menos 2 chamadas à api, uma indicando que a execução teve início (STARTED) e outra sinalizando o fim da execução (SUCCESS) onde é passado o resultado do processamento.

Problema:

A tabela **Execution** deveria ter o campo results com:

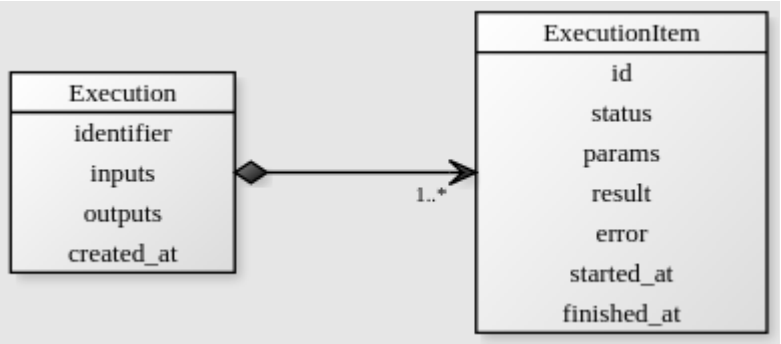
[{'result': 0.95}, {'result': 0.42}, {'result': 0.15}, ...]

Porém, alguns resultados de items não são gravados e ao invés de ter 50 resultados (1 para cada SUCCESS) em uma execução com 50 items, temos 30 ou 40 apenas. Ou seja, a tabela **ExecutionItem** contem os 50 registros com os resultados corretamente, porem a tabela de **Execution** (com relacionamento 1 para N com a ExecutionItem) não tem estes 50 resultados no campo agregador results.

3.1) Qual o problema desta implementação ? Por que isto ocorre?

3.2) Que alteração podemos fazer para garantir que todos os resultados sejam persistidos no banco de dados? De preferência sem alterar o schema do banco de dados (mesmo que não seja o melhor)

4) API MVP de execuções



Dado que temos um banco de dados para armazenar as execuções distribuídas e seus resultados. Precisamos fazer uma API REST para que aplicações clientes possam fazer consultas.

4.1) Quais ferramentas, frameworks e linguagem você utilizaria para fazer esta API de forma rápida, dado que é um MVP para um time interno?

4.2) Pode fazer um pseudo código ou uma implementação parcial das camadas de forma que a versão 0.0.1 da API tenha os seguintes endpoints:

- Retorne uma listagem paginada das execuções
- Retorne detalhes de uma execução específica (todos os items de uma execução)
- Criar uma nova execução

4.3) Pensando que a versão 0.0.1 da sua API ficou pronta e foi passada para o time para utilização, alguns precisam implementar scripts que fazem chamadas na API e outros estão fazendo uma aplicação web para ter uma interface bonitona para exibição dos resultados.

A maior dificuldade reportado pelo time foi que é difícil saber os parâmetros e métodos da API, quais os códigos de retorno e os status http de cada endpoint. Desta forma, eles precisam ficar fazendo chamadas na API para descobrir detalhes e quais os campos são retornados.

Existe alguma maneira para documentar e gerar uma versão 0.0.2 da API que fique fácil saber como utilizar a API?

Em outras palavras, o que você faria para melhorar a comunicação e facilitar a vida de quem precisa utilizar sua API.