Assessment para consultores e desenvolvedores Backend

Histórico da Elaboração do Documento			
Data	Versão	Descrição	Autor
23/03/19	1	Versão Inicial do documento	Cássio Scofield
19/10/19	2	Melhorando critérios de entrega e avaliação	Cássio Scofield
11/11/19	3	Desambiguação	Cássio Scofield

Sumário

1. Visão geral

Esse documento é um exercício que tem como objetivo avaliar o nível de conhecimento e *gaps* a serem preenchidos para profissionais de Consultoria ou Desenvolvimento que trabalham ou desejam trabalhar com backend.

2. Pré-requisitos

É esperado que a pessoa que faça o assessment tenha conhecimento básico em arquitetura cliente-servidor, entenda o funcionamento de requisições HTTP e saiba implementar uma web-API utilizando uma linguagem de programação moderna (Javascript, Java, C#, etc).

Além disso é esperado o respeito à melhores práticas de programação e evitando code-smells sempre que possível.

2.1. Referências

A recomendada a revisão dos seguintes conteúdos antes de começar:

Melhores práticas em Node.JS

https://github.com/i0natan/nodebestpractices

Web-APIs:

https://www.linkedin.com/pulse/m%C3%A1s-pr%C3%A1ticas-em-apis-http-c%C3%A1ssio-scofield

Clean code:

https://youtu.be/9w3o9NHXqu0

Code smells:

https://medium.com/oceanize-geeks/code-smells-and-refactoring-c2c0e0642582

SOLID:

https://en.wikipedia.org/wiki/SOLID

3. Exercício

O exercício consiste em criar o backend de um sistema de marcação e desmarcação de reservas em um clube de tênis.

Timebox esperado de resolução desse exercício: 6 a 18 horas.

Abaixo, seguem as orientações para o desenvolvimento. Lembrando que elas serão a base estrutural para a avaliação.

Por simplicidade, não é necessário conexão com base de dados, os dados podem ser salvos em arquivos ou em memória, todavia é permitido o uso de frameworks de persistência como Loopback, Sequelize, JPA, Hibernate, SpringData, Entity Framework, etc.

3.1. Questão 1 – CRUD, modelo e verbos HTTP

Implementar um CRUD via requisições HTTP no padrão, seguindo uma interface pré-definida.

3.1.1 Criar reserva

Um serviço externo deve conseguir criar uma nova reserva passando como parâmetro a data de início e fim e o tipo de quadra desejada, seguindo a interface abaixo.

```
POST /reservas
Request:
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “inicioEm”: “2018-05-30T18:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,     (string-ISOdate)
}
Response:
statusCode=200
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “inicioEm”: “2018-05-30T18:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,     (string-ISOdate)
  “id”: “20180428924R1L10000”,         (string-ID)
  “status”: “ativa”,                   (string-enum)
  “criadoEm”: “2018-05-29T16:00:00Z”   (string-ISOdate)
}
Os campos em roxo devem ser gerados pela API e retornados para o chamador,
usando valor autogerado para o id, default ("ativa") para o status e dafault
de data atual (new Date()) para o criadoEm, ignorando valores caso inputados
pelo usuário.
```

3.1.2 Buscar reserva por identificador

Um serviço externo deve conseguir consultar os dados da sua reserva a partir do código de reserva.

```
GET /reservas/{id}
```

```
Response:
statusCode=200
body={
  “id”: “20180428924R1L10000”,          (string-ID)
  “tipo”: “SAIBRO”,                     (string-enum)
  “status”: “ativa”,                    (string-enum)
  “criadaEm”: “2018-05-29T18:01:10Z”,   (string-ISOdate)
  “inicioEm”: “2018-05-30T18:00:00Z”,   (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,      (string-ISOdate)
}
```

3.1.3. Alterar reserva

Um serviço externo deve conseguir alterar o tipo de quadra e horário da reserva.

```
PUT /reservas/{id}
Request:
body={
  “tipo”: “SAIBRO”,                     (string-enum)
  “inicioEm”: “2018-05-30T19:00:00Z”,   (string-ISOdate)
  “fimEm”: “2018-05-30T20:00:00Z”,      (string-ISOdate)
}
Response:
statusCode=200
body={
  “id”: “20180428924R1L10000”,          (string-ID)
  “tipo”: “SAIBRO”,                     (string-enum)
  “status”: “ativa”,                    (string-enum)
  “criadaEm”: “2018-05-29T18:01:10Z”,   (string-ISOdate)
  “inicioEm”: “2018-05-30T18:00:00Z”,   (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,      (string-ISOdate)
}
```

3.1.4. Deletar reserva

Um serviço externo deve conseguir cancelar uma reserva a partir do código de reserva.

```
DELETE /reservas/{id}
Response:
statusCode=200
```

3.1.5. Buscar lista de reservas

Um serviço externo deve conseguir consultar todas as reservas.

```
GET /reservas
Response:
statusCode=200
body=[
      {
        “id”: “20180428924R1L10000”,          (string-ID)
        “tipo”: “SAIBRO”,                     (string-enum)
        “status”: “ativa”,                    (string-enum)
        “criadaEm”: “2018-05-29T18:01:10Z”,   (string-ISOdate)
```

```
    "inicioEm": "2018-05-30T18:00:00Z",   (string-ISOdate)
    "fimEm": "2018-05-30T19:00:00Z",      (string-ISOdate)
  }
]
```

3.2. Questão 2 – Regras de negócio

3.2.1. Valores default e calculados

O cliente agora quer que você inclua já o valor e duração da reserva em minutos no email de confirmação, necessitamos adicionar esses campos no retorno da inclusão de reserva.

A duração deve ser o fimEm menos o inicioEm em minutos e o valor deve ser R$0.50 por minuto.

```
POST /reservas
Request:
body={
  "tipo": "SAIBRO",                     (string-enum)
  "inicioEm": "2018-05-30T18:00:00Z",   (string-ISOdate)
  "fimEm": "2018-05-30T20:00:00Z",      (string-ISOdate)
}
Response:
statusCode=200
body={
  "id": "20180428924R1L10000",          (string-ID)
  "tipo": "SAIBRO",                     (string-enum)
  "status": "ativa",                    (string-enum)
  "criadaEm": "2018-05-29T18:01:10Z",   (string-ISOdate)
  "inicioEm": "2018-05-30T18:00:00Z",   (string-ISOdate)
  "fimEm": "2018-05-30T19:00:00Z",      (string-ISOdate)
  "duracao": 120                        (int),
  "valor": 60.00                        (float)
}
```

3.2.2. Deleção lógica e override

Não obstante, ele nos notificou que muitos clientes estão reclamando que suas reservas estão desaparecendo do sistema do nada e, por isso, precisamos adicionar uma forma para os usuários verem o histórico também dos cancelamentos.

Precisamos então fazer o cancelamento lógico das reservas ao invés de apagá-las da base de dados para sempre e adicionar a data de cancelamento na interface.

```
DELETE /reservas/{id}
Response:
statusCode=200
body={
  "id": "20180428924R1L10000",          (string-ID)
```

```
  “tipo”: “SAIBRO”,                    (string-enum)
  “status”: “cancelada”,               (string-enum)
  “criadaEm”: “2018-05-29T18:01:10Z”,  (string-ISOdate)
  “inicioEm”: “2018-05-30T18:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,     (string-ISOdate)
  “duracao”: 60,                       (int)
  “valor”: 30.00,                      (float)
  “canceladaEm”: “2018-05-30T19:00:00Z” (string-ISOdate)
}
```

3.3. Questão 3 – Validação de formatos e regras simples

Percebemos que apareceram uns "lixos" no campo de "status" e algumas horas quebradas no campo "duracao" na base de dados porque não havia validação dos campos de entrada e desenvolvedor do front-end, que era muito junior e não fez a validação do lado de lá.

3.3.1. Validação enums

Precisamos validar a consistência dos dados antes de salvá-los para evitar problemas no futuro, respondendo com statusCode adequado quando a requisição for inválida.

Adicionar a regra para validar o campo status; este deve conter apenas os valores "ativo", "cancelado" e "pago".

```
PUT /reservas/{id}
Request:
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “inicioEm”: “2018-05-30T19:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T20:00:00Z”,     (string-ISOdate)
  “status”: “deletado”,                (string-enum)
}
Response:
statusCode=400
```

3.3.2. Validação de regras de negócio simples

Precisamos validar se o "duracao" da reserva está sempre maior que 60 minutos, pois o clube decidiu que não é viável. Para facilitar a operação foi decidido que todas as reservas devem ter "duracao" múltiplos 60 minutos, sempre começando nos inícios de horas, não podendo uma reserva começar ou terminar em horas quebradas. As reservas podem ser múltiplos de 1h (1h, 2h, 3h), mas não podem ser horas quebradas (1h30min).

```
POST /reservas
Request:
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “inicioEm”: “2018-05-30T18:30:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,     (string-ISOdate)
```

```
}
Response:
statusCode=422
```

3.4. Questão 4 -Regra de negócio

O sistema de reservas ficou popular e agora quase todo o clube está usando, mas infelizmente estamos reservando a mesma quadra para mais de uma pessoas e os clientes estão muito irritados com isso.

Não podemos deixar dois usuários reservarem a mesma quadra no mesmo horário, sendo assim, antes de criar uma reserva precisamos verificar se aquele "slot" já foi usado em uma reserva ativa.

O clube possui apenas 2 quadras, uma de SAIBRO e uma HARD e não tem pretensão de expansão no futuro próximo.

3.3.1. Validação de disponibilidade

Antes de gravar a nova reserva precisamos conferir se o range (inicio - fim) já não foi reservado anteriormente e em caso afirmativo, devemos informar ao usuário que não podemos fazer a reserva.

```
POST /reservas
Request:
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “inicioEm”: “2018-05-30T18:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T19:00:00Z”,     (string-ISOdate)
}
Response:
statusCode=422
body={
  "error": {
    "message": "O horário solicitado não está disponível, favor
selecione um outro horário.",
    "code": "HORARIO_INDISPONIVEL"
  }
}
```

3.5. Questão 5 - Rotas customizadas

Implementamos a regra para evitar overbooking, mas a usabilidade ficou muito ruim porque o usuário tem que fazer a reserva por tentativa e erro. Sendo assim, o nosso cliente pediu para que pudéssemos listar para o cliente os horários disponíveis para a quadra que ele deseja na data que ele deseja.

3.3.2. Consulta de disponibilidade

Vamos começar pela regra mais simples: Caso o horário/tipo solicitado esteja livre, retorna-o.

```
POST /disponibilidade
Request:
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “inicioEm”: “2018-05-30T19:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T20:00:00Z”,     (string-ISOdate)
}
Response:
body=[{
  “tipo”: “SAIBRO”,                    (string-enum)
  “duracao”: 60,                       (int)
  “inicioEm”: “2018-05-30T19:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T20:00:00Z”,     (string-ISOdate)
}]
```

3.3.2. Consulta de disponibilidades similares

Caso o horário/tipo solicitado não esteja disponível, retorno uma lista de disponibilidades semelhantes ainda disponível. Seguindo a preferência:

3.3.2.1. Mesmo horário, outro tipo de quadra
3.3.2.2. Mesmo tipo de quadra, uma hora antes e/ou depois
3.3.2.3. Outro tipo de quadra, uma hora antes e/ou depois
3.3.2.4. Mesmo tipo de quadra, duas hora antes e/ou depois
3.3.2.5. Caso nenhuma das regras se encaixe o retorno é vazio.

Obs: por simplicidade, não vamos nos preocupar agora com reservas de mais de uma hora.

```
GET /disponibilidade
Request:
body={
  “tipo”: “SAIBRO”,                    (string-enum)
  “duracao”: 60,                       (int)
  “inicioEm”: “2018-05-30T19:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T20:00:00Z”,     (string-ISOdate)
}
Response:
body=[{
  “tipo”: “HARD”,                      (string-enum)
  “duracao”: 60,                       (int)
  “inicioEm”: “2018-05-30T19:00:00Z”,  (string-ISOdate)
  “fimEm”: “2018-05-30T20:00:00Z”,     (string-ISOdate)
}, {
  “tipo”: “SAIBRO”,                    (string-enum)
  “duracao”: 60,                       (int)
```

```
    “inicioEm”: “2018-05-30T18:00:00Z”,   (string-ISOdate)
    “fimEm”: “2018-05-30T19:00:00Z”,      (string-ISOdate)
  }, {
    “tipo”: “SAIBRO”,                     (string-enum)
    “duracao”: 60,                        (int)
    “inicioEm”: “2018-05-30T21:00:00Z”,   (string-ISOdate)
    “fimEm”: “2018-05-30T22:00:00Z”,      (string-ISOdate)
  }, {
    “tipo”: “HARD”,                       (string-enum)
    “duracao”: 60,                        (int)
    “inicioEm”: “2018-05-30T21:00:00Z”,   (string-ISOdate)
    “fimEm”: “2018-05-30T22:00:00Z”,      (string-ISOdate)
  }]
```

4. Entrega

Para entrega do exercício, criem um repositório git público no github ou ibmcloud e dêem acesso ao avaliador, contendo um readme com passo a passo para instalação e configuração de qualquer ferramenta que seja necessária para rodar o app e enviem a URL do repositório por email.

5. Avaliação

A avaliação será feita dividida em 3 partes com score máximo de 100 pontos.

Funcional: 50 pontos será avaliado objetivamente a partir de casos de teste pré definidos em um script de teste, sendo o resultado proporcional ao número de casos de teste que passaram. Todos os casos de teste são chamadas direto aos métodos da API via postman e serão avaliados em relação ao formato da requisição (URL, parâmetros e método) e resultado retornado (body e status code), portanto não há necessidade de criar interface para chamada dos métodos.

Não funcionais: 50 pontos será avaliado na inspeção manual do código para verificar aspectos não funcionais:

- Documentação: Adicionar um README.md descrevendo como rodar a aplicação, passo a passo, contendo pré-requisitos.

- Melhores práticas: Nome dos métodos, classes, responsabilidades

- Code smells: Muitos code smells? Nível de gambiarra baixo?

- Manutenibilidade/Debugabilidade: Uso de ferramentas de mercado, uso de mensagens de erro que façam sentido. Há logs de erros? pode ser console.log mesmo; ofuscar erros nunca.

- Performance - Apareceu algum gargalo muito óbvio ou algum problema com a complexidade ciclomática de algum dos endpoints?

WOW factor: 20 pontos extra para criação de requisitos novos, principalmente, mas não somente não funcionais e bem explicados/documentados, que façam sentido, estes devem estar descrito no README.md. É importante todavia lembrar que as interfaces definidas nos itens principais devem ser respeitadas, caso elas sejam alteradas ou removida para tentar fazer uma funcionalidade extra não solicitada, pontos de completude serão subtraídos.

Ex: Testes unitários, roteiro de testes próprio, testes end-to-end, autenticação e/ou autorização, resultado de um teste de carga, deploy na nuvem, propor e implementar nova regras mais inteligente de busca de disponibilidade.

O resultado da avaliação será enviado por email com o detalhamento dos casos de teste errados e os itens não funcionais e GAPs a melhorar.

Boa sorte!