PUC Minas
DIRETORIA DE
EDUCAÇÃO CONTINUADA

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO DIRETORIA DE EDUCAÇÃO CONTINUADA

Curso/Oferta: Arquitetura de Software Distribuído – Oferta 12
Disciplina: Arquitetura de Backend e Microsserviços
Professor: Marco Mendes
Alunos: Eduardo Henrique Borges da Silva / João Bosco Calais Neto
Data: 10/09/19

Justificativa atividade aula 03

1. Organize APIs ao longo de recursos. As URIs expostos por um serviço REST devem ser baseados em substantivos (os dados aos quais a API da Web fornece acesso) e não em verbos (o que uma aplicação pode fazer com os dados).

```
// Get comment by book
app.get('/v1/public/comments/:bookId', function(req, res) {
    var commentsByBook = [];
    for(var i = 0; i < comments.length; i++){
        if(req.params.bookId == comments[i].book.id){
            commentsByBook.push(comments[i]);
        }
    }
    res.status(200).send(JSON.stringify(commentsByBook));
});
```

2. Padronize as suas APIs. Adote uma convenção de nomenclatura consistente nas URIs. Em geral, é útil usar substantivos no plural para URIs que fazem referência a coleções.

```
// Delete book
app.delete('/v1/public/books/:id', function(req, res) {
    var index;
    for(var i = 0; i < books.length; i++){
        if(req.params.id == books[i].id){
            index = i;
            break;
        }
    }
    books.splice(index, 1);
    res.status(200);
});

/*=============================================================================

Requests

=============================================================================*/

// Get all requests
app.get('/v1/public/requests', function(req, res) {
    res.status(200).send(JSON.stringify(requests));
});
```

3. Evite APIs anêmicas. Evite projetar uma interface REST que espelhe ou dependa da estrutura interna dos dados que ela expõe. O REST é mais do que implementar operações CRUD simples (Create, Retrieve, Update, Delete) sobre tabelas separadas em um banco de dados relacional. O objetivo do REST é mapear as entidades de negócios e as operações que um aplicativo pode executar nessas entidades para a implementação física dessas entidades, mas um cliente não deve ser exposto a esses detalhes físicos.

```
// Get all books
app.get('/v1/public/books', function(req, res) {
    res.status(200).send(JSON.stringify(books));
});

// Register book
app.post('/v1/public/books', function(req, res) {
    books.push(req.body);
    res.status(201).send(JSON.stringify(req.body));
});

// Get book by id
app.get('/v1/public/books/:id', function(req, res) {
    var selectedBook = {};
    for(var i = 0; i < books.length; i++){
        if(req.params.id == books[i].id){
            selectedBook.push(books[i]);
            break;
        }
```

4. Crie APIs simples. Evite criar URIs de recursos mais complexos do que coleção/item/coleção.

```
// Register book
app.post('/v1/public/books', function(req, res) {
    books.push(req.body);
    res.status(201).send(JSON.stringify(req.body));
});
```

5. Considere a atualização em lote para operações complexas. Considere implementar operações HTTP PUT em massa que possam atualizar em lotes vários recursos de uma coleção de dados. A solicitação PUT deve especificar o URI da coleção e o corpo de solicitação deve especificar os detalhes dos recursos a serem modificados. Esta abordagem pode ajudar a reduzir ineficiências do protocolo HTTP e melhorar o desempenho da sua aplicação.

```
// Edit comment
app.put('/v1/public/comments/:id', function(req, res) {
    var updateReview = {};
    for(var i = 0; i < comments.length; i++){
        if(req.params.id == comments[i].id){
            updateReview = comments[i];
            break;
        }
    }
    updateReview.content = req.body;
    res.status(200).send(JSON.stringify(updateReview));
});
```

6. Se você precisar receber datas e horas nas API, use o padrão ISO 8601. APIs que trafegam datas ou horas devem usar o padrão ISO 8601 para garantir interoperabilidade - https://www.w3.org/TR/NOTE-datetime.

Não foi necessário.

7. Documente sua API. Uma API é tão boa quanto sua documentação. Os documentos devem ser fáceis de encontrar e publicamente acessíveis. A maioria dos desenvolvedores verificará os documentos antes de tentar qualquer esforço de integração. Quando os documentos estão ocultos, ausentes ou obscuros, qualquer esforço de integração será muito aumentado.

GET	/v1/public/comments	Get all comments
GET	/v1/public/comments/{bookId}	Get comment by book
POST	/v1/public/comments/{bookId}	Register comment by book
GET	/v1/public/comments/{id}	Get comment by id
PUT	/v1/public/comments/{id}	Edit comment
DELETE	/v1/public/comments/{id}	Delete comment

8. Use protocolo HTTPS/SSL. Sempre! Suas APIs da web podem ser acessadas de qualquer lugar onde haja internet (como bibliotecas, cafés, aeroportos, entre outros). Nem todos estes são seguros. Muitos não criptografam as comunicações, permitindo fácil escutas ou falsificação de identidade se as credenciais de autenticação forem sequestradas. Outra vantagem de sempre usar o SSL é que as comunicações criptografadas garantidas simplificam os esforços de autenticação - você pode usar tokens de acesso simples em vez de assinar cada solicitação da API.

Não foi necessário.

9. Versione suas APIs. O versionamento permite que você possa evoluir suas APIs sem quebrar o funcionamento de seus clientes. Se você examinar cada URL da API da Marvel verá que a informação de versão foi disponibilizada.

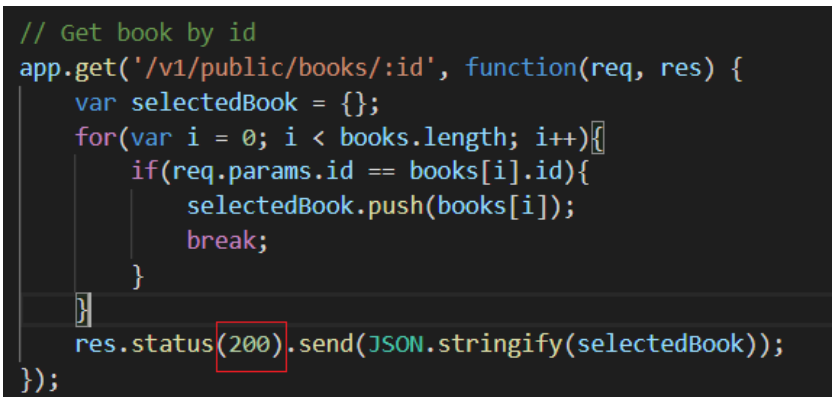
```
// Edit book
app.put('/v1/public/books/:id', function(req, res) {
    var updateBook = {};
    for(var i = 0; i < books.length; i++){
        if(req.params.id == books[i].id){
            updateBook = books[i];
            break;
        }
    }
    updateBook = req.body;
    res.status(200).send(JSON.stringify(updateBook));
});

// Delete book
app.delete('/v1/public/books/:id', function(req, res) {
    var index;
    for(var i = 0; i < books.length; i++){
        if(req.params.id == books[i].id){
            index = i;
            break;
        }
    }
```

10. Estabeleça paginação para coleções com grandes volumes de dados. Não faz sentido retornar 20.000 clientes em um único pacote JSON. E para evitar grandes pacotes de dados devemos estabelecer controles de paginações. Veja por exemplo os dois parâmetros abaixo da API de listagem de personagens da Marvel (limit e offset), que controlam a quantidade de dados retornados e a posição de entrada.

    Não foi necessário.

11. Use corretamente os códigos de retorno HTTP.



```
// Get book by id
app.get('/v1/public/books/:id', function(req, res) {
    var selectedBook = {};
    for(var i = 0; i < books.length; i++){
        if(req.params.id == books[i].id){
            selectedBook.push(books[i]);
            break;
        }
    }
    res.status(200).send(JSON.stringify(selectedBook));
});
```