Universidade São Judas Tadeu

Sistemas Distribuídos e Mobile

1 Introdução

Como vimos, um dos principais protocolos da camada de aplicação é o HTTP. Neste material, estudaremos mais detalhes sobre o seu funcionamento implementando um novo servidor com NodeJS e utilizando um cliente HTTP como o Postman. Lembre-se que o protocolo HTTP (assim como muitos outros) é definido em um documento chamado RFC (Request for Comment). A RFC 7231 em que o HTTP é definido pode ser encontrada no Link 1.1.

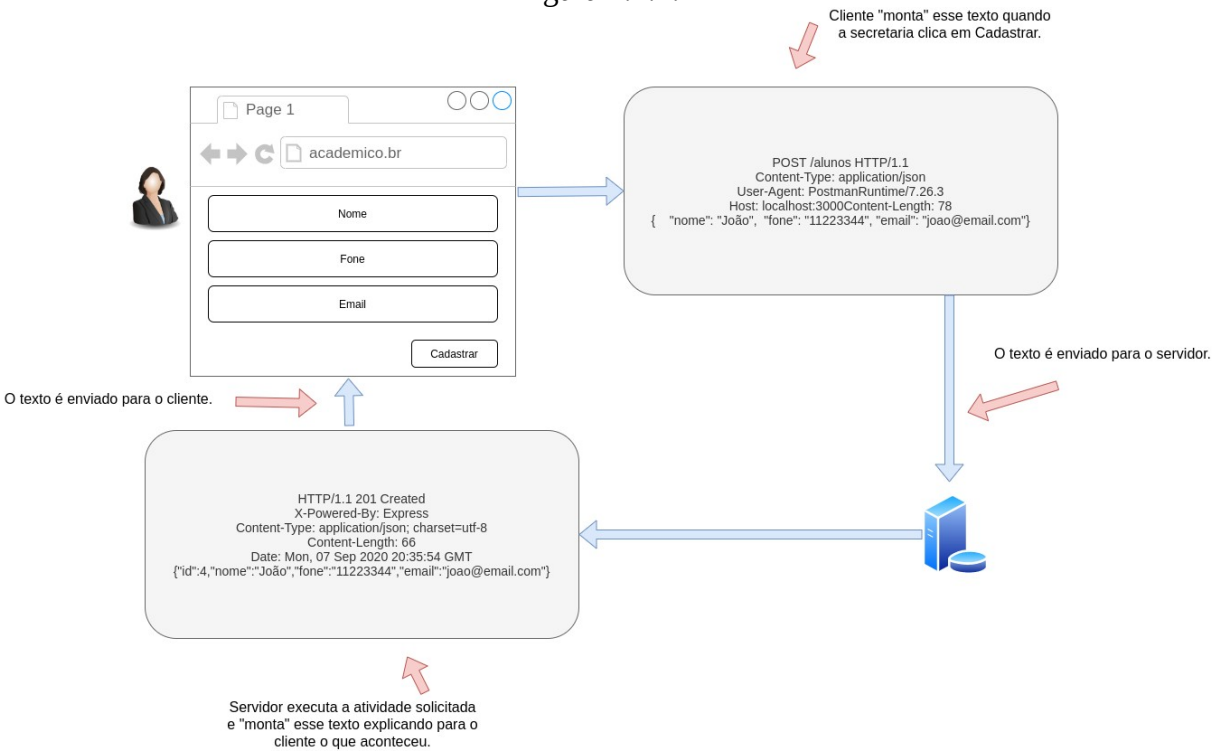
Link 1.1
https://tools.ietf.org/html/rfc7231

2 Passo a passo

2.1 (Simulação gráfica) Considere o sistema acadêmico de uma instituição. Ele é responsável por gerenciar dados de alunos, professores, notas, cursos, disciplinas entre muitas outras coisas. Um sistema assim pode ter uma interface gráfica própria ou pode ser que ele seja utilizado por meio de um navegador comum. Independentemente disso, nos dias atuais, é muito comum que a troca de informações entre cliente (a interface gráfica ou no navegador, por exemplo) e servidor ocorra utilizando o protocolo HTTP. Veja as figuras a seguir.

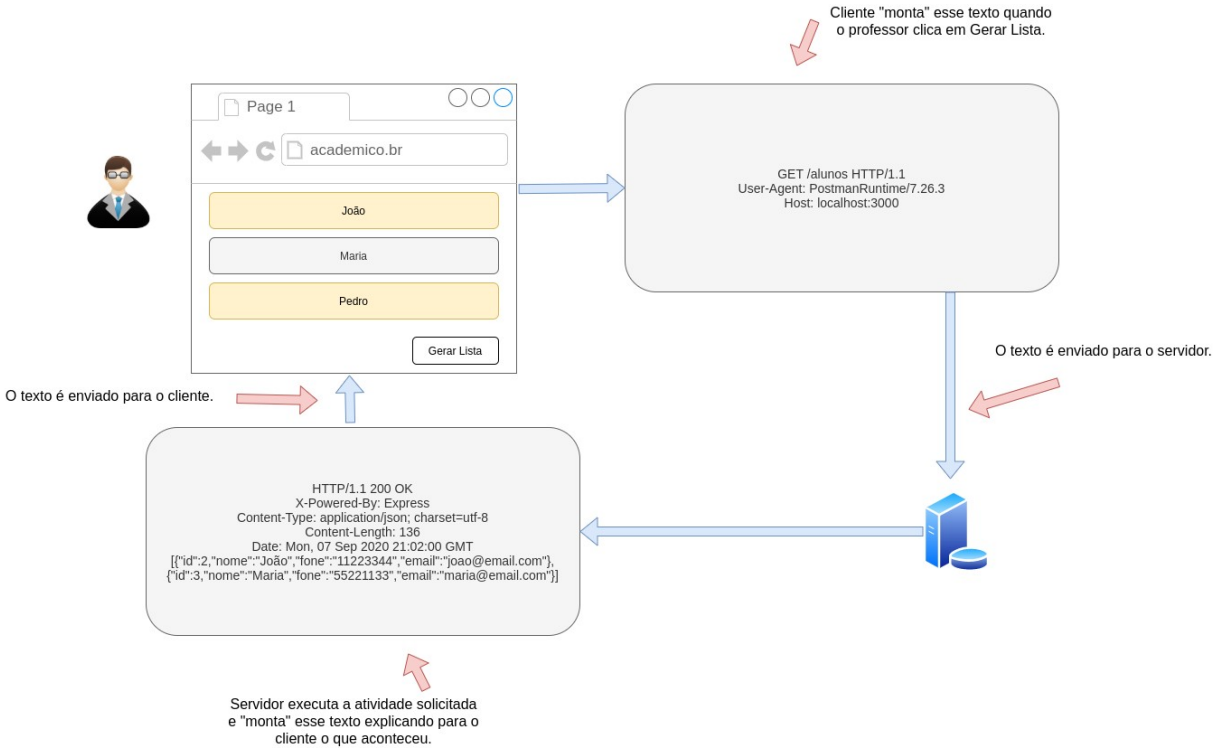
2.1.1 (Cadastro de aluno) Considere um novo aluno que deseja se matricular. Ele visitará a secretaria e a atendente o ajudará com isso. Ela acessará o sistema utilizando o software cliente (o navegador, neste exemplo) e, por meio de controles visuais (botões, campos textuais etc) irá interagir com o servidor, explicando a sua necessidade. Lembre-se que o protocolo HTTP define diversos **métodos** ou **verbos**, cada qual sua própria semântica e casos de uso apropriados. Para este caso de uso, o método **POST** é uma boa opção. Assim, uma vez que a secretaria clicar no botão para confirmar o cadastro do novo aluno, o software cliente irá construir uma mensagem HTTP (completamente textual) e enviar para o software servidor, responsável por interpretá-la e executar as funções solicitadas. Uma vez que terminar o processamento, o software servidor deverá explicar para o cliente o que aconteceu, por meio do envio de uma mensagem que também está de acordo com a especificação do protocolo HTTP. Ele vai, por exemplo, entregar para o cliente um **código de status** que indica o que houve. Veja a Figura 2.1.1.1.

Figura 2.1.1.1



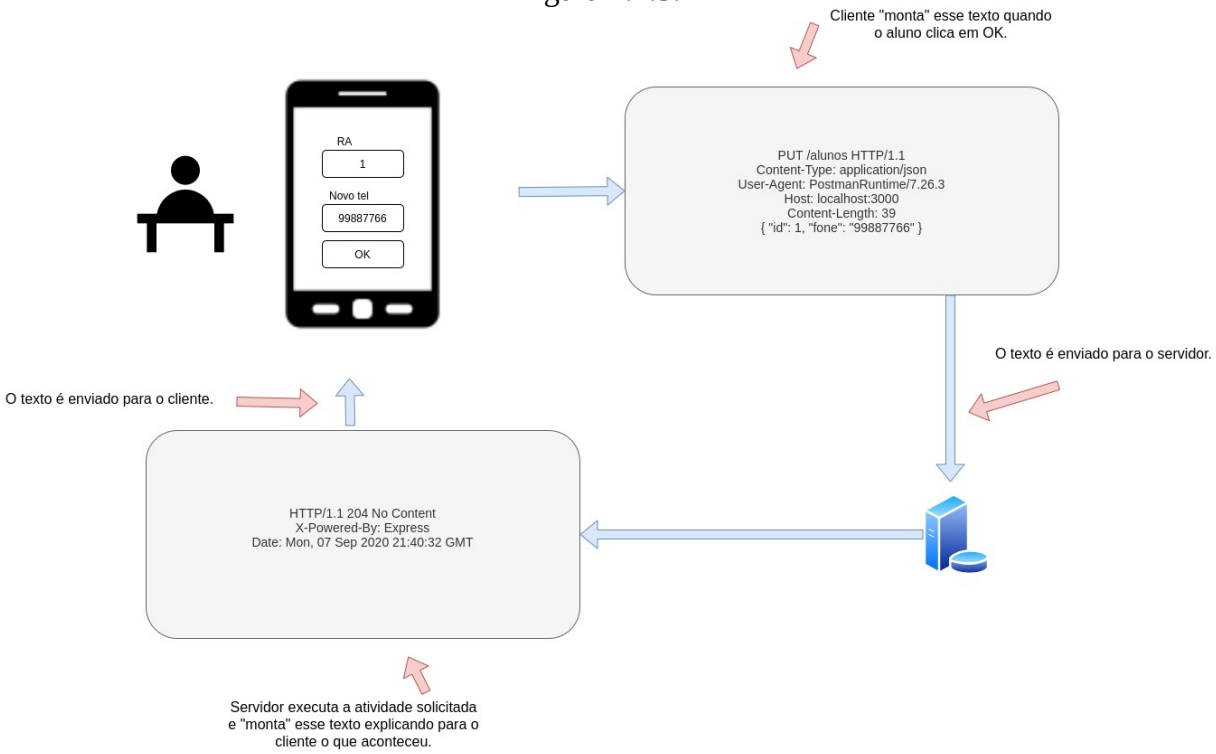
2.1.2 (Listagem de alunos) Um outro caso de uso típico é a listagem de alunos. Um professor, também usuário do sistema, pode estar interessado em obter uma lista com os nomes de seus alunos. Para este cenário, o método **GET** do protocolo HTTP pode ser uma boa opção. Por sua vez, o servidor pode responder algo como **200 OK**, que é um código de resultado que indica sucesso e que, para requisições GET, indica que o conteúdo solicitado pode ser encontrado no corpo da resposta. Veja a Figura 2.1.2.1.

Figura 2.1.2.1



2.1.3 (Atualização de cadastro) Um caso de uso muito típico envolve atualizações cadastrais. Um nome cadastrado errado, uma alteração de endereço ou telefone entre muitas outras coisas podem fazer com que o aluno entre em contato com a instituição para solicitar a atualização. Pode ser também que a instituição disponibilize um aplicativo para dispositivos móveis que permite que o próprio aluno faça a atualização de alguns campos de seu cadastro. Suponha que o aluno deseja utilizar o aplicativo para atualizar o seu telefone. Utilizando a interface gráfica do aplicativo, ele informa seu registro acadêmico e o novo telefone. Note que o cliente agora é um aplicativo, porém o protocolo para comunicação com o servidor permanece o mesmo: o HTTP. Atualizações, em geral, são feitas utilizando-se o método **PUT**. O servidor, por sua vez, caso deseje apenas informar se a operação foi realizada com sucesso ou não sem enviar nenhum outro dado (a lista completa de alunos, por exemplo, seria desnecessária) pode responder algo como **204 No Content**. Veja a Figura 2.1.3.1.

Figura 2.1.3.1


Cliente "monta" esse texto quando o aluno clica em OK.
RA
1
Novo tel
99887766
OK
PUT /alunos HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.26.3
Host: localhost:3000
Content-Length: 39
{ "id": 1, "fone": "99887766" }
O texto é enviado para o servidor.
O texto é enviado para o cliente.
HTTP/1.1 204 No Content
X-Powered-By: Express
Date: Mon, 07 Sep 2020 21:40:32 GMT
Servidor executa a atividade solicitada e "monta" esse texto explicando para o cliente o que aconteceu.

2.2 (Implementação do servidor) Nesta seção iremos implementar, utilizando o NodeJS, o servidor que disponibiliza as funcionalidades descritas, além de testá-lo com o Postman.

- Comece criando um diretório para você trabalhar. A seguir, abra um terminal (prompt de comando) e navegue até o seu diretório com

cd diretorio

- Para criar um novo projeto NodeJS, podemos usar o **N**ode **P**ackage **M**anager (NPM). Para isso, use

npm init

A seguir, no terminal, você deverá responder algumas perguntas como qual será o nome do projeto e a sua versão. Mantenha todos os valores padrão pressionando Enter.

- Você pode utilizar o IDE que desejar. Neste material, utilizamos o VS Code, um dos IDEs mais utilizados no momento. Para abrir uma instância dele já vinculada ao diretório atual, use

code .

Você também pode abrir o VS Code usando o atalho gráfico do seu sistema operacional e então, escolher a opção File >> Open Folder.

- Para implementar o nosso servidor, utilizaremos os seguintes pacotes

- **express** - framework para aplicações Node que simplifica a manipulação de requisições HTTP.
- **body-parser** – pacote que simplifica a manipulação de objetos no formato JSON, por exemplo
- **nodemon** – O nodemon (de nodemon) faz **live reload** de nosso servidor para que não seja necessário reiniciar o servidor a cada nova atualização.

- No VS Code, clique em Terminal >> New Terminal (ou você também pode usar o terminal que havia aberto anteriormente) e instale os pacotes da seguinte forma

npm install --save express
npm install --save body-parser
npm install --save-dev nodemon

- Nosso servidor é muito simples. Ele terá somente as funcionalides que descrevemos anteriormente e manipulará uma base em memória volátil. Por isso, um único arquivo é suficiente para a sua implementação. No Vs Code, crie uma nova pasta chamada **src** (ela deve ficar na raiz do projeto, lado a lado com o arquivo **package.json**) e, dentro da pasta **src**, crie um arquivo chamado **index.js.**

- Para colocar o servidor em execução, vamos criar um script no arquivo **package.json**. Um script nada mais é do que um apelido que atribuímos a um comando qualquer para que seja mais fácil executá-lo. Abra o arquivo package.json e adicione a linha destacada na Listagem 2.2.1.

Listagem 2.2.1

```
{
  "name": "pessoal_basico_http_crud_alunos",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon src/index.js --exec node"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    "nodemon": "^2.0.4"
  }
}
```

- Coloque o servidor em execução com

npm run start

- Para começar a implementação do servidor, precisamos trazer para o contexto os pacotes necessários e fazer especificações iniciais, como a porta a que ele estará vinculado. Isso pode ser feito como mostra a Listagem 2.2.2. Atente-se aos comentários no código.

Listagem 2.2.2

```
//importa os módulos http e express
const http = require('http');
const express = require('express');
//constrói um objeto express
const app = express();
//importa o body-parser
const bodyParser = require('body-parser');
app.use(bodyParser.json());

//configura a porta do servidor e o coloca em execução.
const porta = 3000;
app.set('port', porta);
const server = http.createServer(app);
server.listen(3000);
```

- A seguir, vamos definir a coleção de dados que representará os alunos da instituição. Veja a Listagem 2.2.3.

Listagem 2.2.3

```
let id = 1;
let alunos = [
  {
    id: 1,
    nome: "João",
    fone: "11223344",
    email: "joao@email.com"
  },
  {
    id: 2,
    nome: "Maria",
    fone: "55221133",
    email: "maria@email.com"
  }
];
```

- Resta especificar o que ocorre quando requisições do tipo **POST**, **GET** e **PUT** são recebidas no endpoint **/alunos**. Para isso, usamos o objeto **app** construído anteriormente. Ele dispõe de métodos apropriadoas para cada verbo HTTP. Veja a Listagem 2.2.4.

Listagem 2.2.4

```
//tratamento de requisições POST
app.post ("/alunos", (req, res, next) => {
  const aluno = {
    id: id +=1,
    nome: req.body.nome,
    fone: req.body.fone,
    email: req.body.email
  }
  alunos.push (aluno)
  res.status(201).json(aluno);
});

//tratamento de requisições GET
app.get ("/alunos", (req, res, next) => {
  res.status(200).json(alunos);
})

//tratamento de requisições PUT
app.put("/alunos", (req, res, next) => {
  alunos.forEach ((aluno) => {
    if (aluno.id === req.body.id){
      aluno.fone = req.body.fone
    }
  })
  res.status(204).end();
});
```

- Estude o código da Listagem 2.2.5 e compare com a simulação gráfica.

- Cada operação pode ser testada utilizando-se um cliente HTTP, como o Postman. Veja como utilizá-lo nas de 2.2.5 a 2.2.5. Seu uso é muito simples. Escolhemos o verbo HTTP, o endpoint e, se for o caso, especificamos os dados a serem enviados para o servidor. Na parte inferior, no console, podemos ver as requisições e respostas textualmente.

Figura 2.2.5

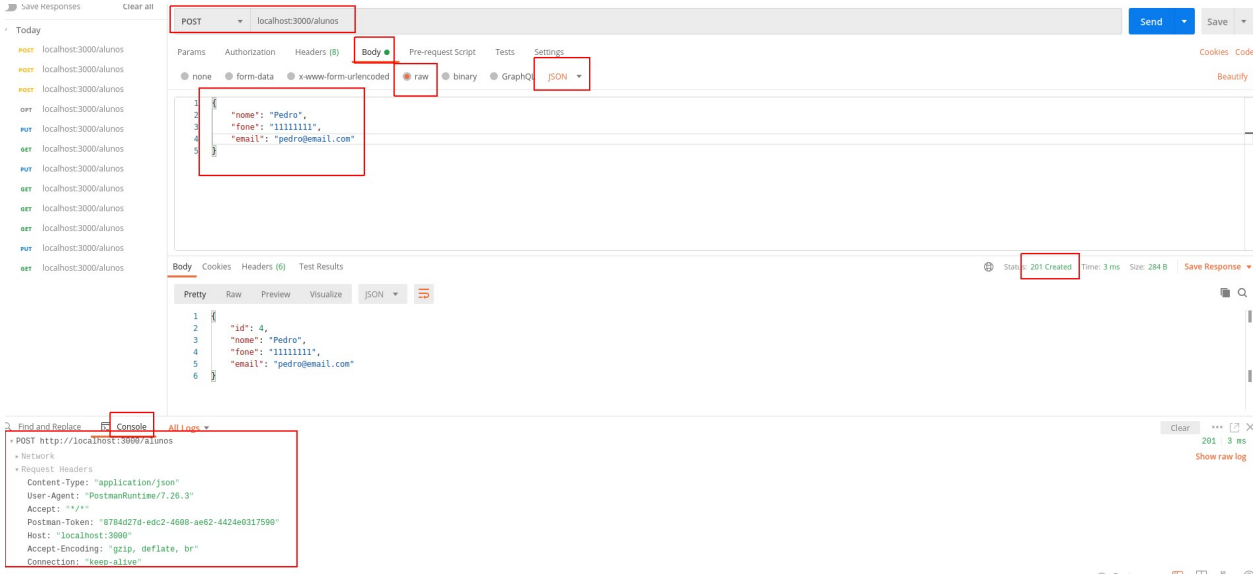


Figura 2.2.6

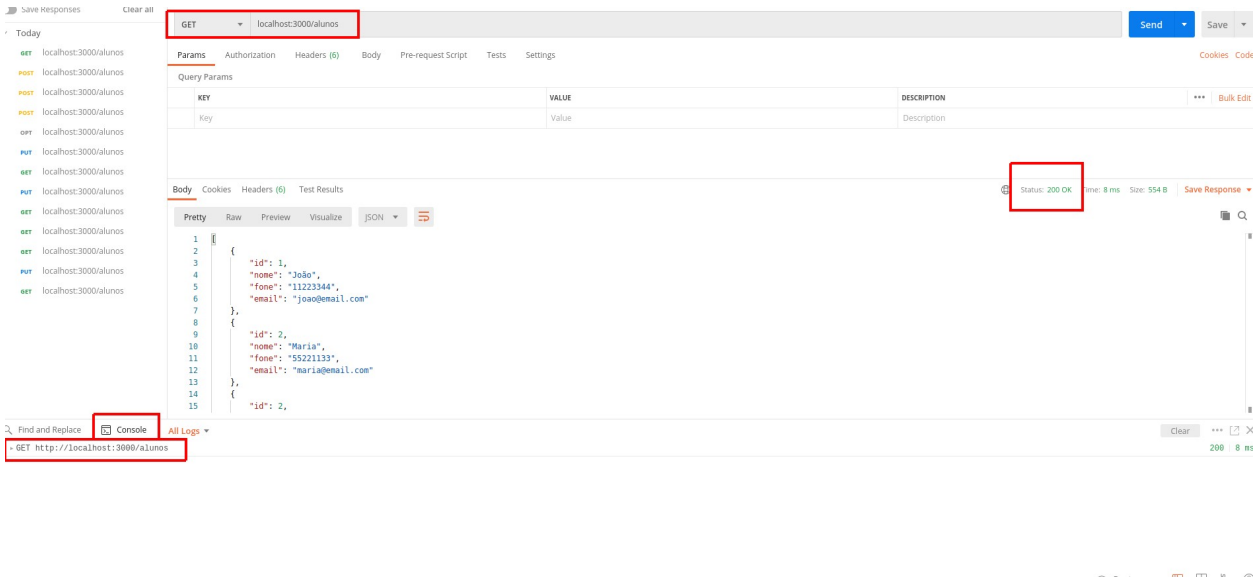
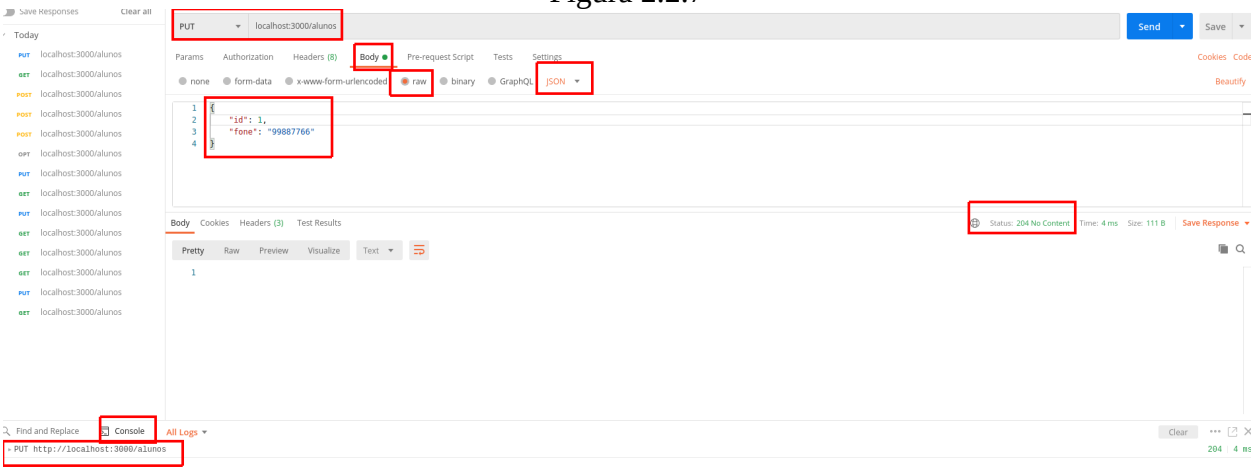


Figura 2.2.7


Save Responses
Clear all
Today
PUT localhost:3000/alunos
GET localhost:3000/alunos
POST localhost:3000/alunos
POST localhost:3000/alunos
POST localhost:3000/alunos
OPT localhost:3000/alunos
PUT localhost:3000/alunos
GET localhost:3000/alunos
PUT localhost:3000/alunos
GET localhost:3000/alunos
GET localhost:3000/alunos
GET localhost:3000/alunos
PUT localhost:3000/alunos
GET localhost:3000/alunos
PUT
localhost:3000/alunos
Send
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Code
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id": 1,
"fone": "99887766"
}
Body
Cookies
Headers (3)
Test Results
Status: 204 No Content
Time: 4 ms
Size: 111 B
Save Response
Pretty
Raw
Preview
Visualize
Text
Find and Replace
Console
All Logs
Clear
PUT http://localhost:3000/alunos
204
4 ms

Exercícios

1. Implemente um método no servidor que permita que um aluno seja removido da base. Estude a especificação HTTP e escolha o verbo e o código de resposta mais apropriados.

2. Implemente um CRUD de livros. Livros têm id, titulo, descricao, edicao e autor. O servidor deve utilizar os verbos POST, GET, PUT e DELETE para cada operação. Para cada uma, escolha um código de status apropriado. Em particular, para a funcionalidade que remove livros, devolva como resultado a lista de livros, além é claro de um código de status apropriado.

Referências

MDN Web Docs. 2020. Disponível em <https://developer.mozilla.org/en-US/>. Acesso em agosto de 2020.

Node.js. 2020. Disponível em <https://nodejs.org>. Acesso em agosto de 2020.

Postman | The Collaboration Platform for API Development. 2020. Disponível em <https://www.postman.com/>. Acesso em agosto de 2020.

RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. 2020. Disponível em <https://tools.ietf.org/html/rfc2616>. Acesso em agosto de 2020.

RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. 2020. Disponível em <https://tools.ietf.org/html/rfc7231#section-6.3.3>. Acesso em setembro de 2020.