Laboratório 5

Sistemas Distribuídos
Prof Adailton de Jesus Cerqueira Junior

Neste laboratório, daremos continuidade à criação de uma aplicação simples utilizando NodeJS com o framework Express. Agora, o foco será explorar outros verbos do protocolo HTTP, como POST e PUT.

Servidor:

1 – Crie uma pasta chamada lab05 e dentro desta pasta vamos criar nosso projeto com o comando *npm init*.

No terminal, serão feitas algumas perguntas sobre o projeto. Vamos responder às perguntas (podemos deixar o valor default) e ao finalizar será criado um arquivo chamado **package.json** com as configurações do projeto.

```
{
  "name": "jogo-api",
  "version": "1.0.0",
  "description": "API para listar jogos",
  "main": "index.js",
  ▷Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

2 – Agora vamos instalar o pacote Express com o comando *npm install express*.

3 – Copie o arquivo **index.js** do laboratório 04 e, no terminal, inicie o servidor.

DICA: Lembre-se de instalar o *nodemon* para auxiliar no desenvolvimento.

Persistência:

1 – Vamos modificar nosso servidor para persistir as informações referentes aos registros de jogos. Para isso, vamos utilizar a biblioteca ***fs***.

```
const express = require('express');
const fs = require('fs');
```

2 – Com a biblioteca já adicionada, vamos criar um arquivo responsável por armazenar os registros dos jogos, conforme o exemplo de código a seguir.

```
const app = express();
const arquivo = 'jogos.db';

app.listen(3000, () => {
  console.log('API de jogo em execução na porta 3000.');
  console.log('Acesse a url http://localhost:3000');

  fs.access(arquivo, fs.constants.F_OK, (err) => {
    if (err) {
      console.log(`${arquivo} não existe. Criando arquivo...`);
      let jogosIniciais = [
        { id: 1, nome: "Super Mario World", ano: 1990, categoria: "Plataforma" },
        { id: 2, nome: "Age of Empires II", ano: 1999, categoria: "Estratégia" },
        { id: 3, nome: "The Elder Scrolls V: Skyrim", ano: 2011, categoria: "RPG" },
        { id: 4, nome: "The Last of Us", ano: 2013, categoria: "Aventura" },
        { id: 5, nome: "God of War", ano: 2018, categoria: "Ação" }
      ];
      fs.writeFileSync(arquivo, JSON.stringify(jogosIniciais));
    }
  });
});
```

3 – Precisamos atualizar o endpoint que retorna os jogos cadastrados na aplicação.

```
app.get('/jogos', (req, res) => {
  let data = fs.readFileSync(arquivo);
  let jogos = JSON.parse(data);

  // Verificando se foi passado um parâmetro de busca
  if (req.query.categoria) {
    jogos = jogos.filter(jogo => jogo.categoria.toLowerCase()
      .includes(req.query.categoria.toLowerCase()));
  }

  res.send(jogos);
});
```

4 – Aproveitando, vamos criar um novo endpoint utilizando o verbo GET para retornar um registro com base no id informado.

```
app.get('/jogos/:id', (req, res) => {
  let data = fs.readFileSync(arquivo);
  let jogos = JSON.parse(data);
  let jogo = jogos.find(jogo => jogo.id == req.params.id);

  if (jogo) {
    res.send(jogo);
  } else {
    res.status(404).send('Jogo não encontrado.');
  }
});
```

Observe que utilizamos um parâmetro dinâmico na rota. Nesse caso, o id representa o identificador do registro que será recuperado.

Rotas:

1 – Agora, vamos criar uma rota para salvar novos registros de jogos. Para isso, utilizaremos o verbo POST com o endpoint */jogos*, conforme demonstrado a seguir.

```
app.post('/jogos', (req, res) => {
  let data = fs.readFileSync(arquivo);
  let jogos = JSON.parse(data);
  let novoJogo = req.body;

  novoJogo.id = jogos.length + 1;
  jogos.push(novoJogo);

  fs.writeFileSync(arquivo, JSON.stringify(jogos));
  res.status(201).send(novoJogo);
});
```

2 – Antes de realizar algum teste no endpoint com o verbo POST, devemos colocar uma instrução para realizar o parse do body da requisição.

```
// Realiza um parse do body para uma estrutura JSON
app.use(express.json());
```

3 – Agora, vamos utilizar o verbo PUT para editar um registro.

```
app.put('/jogos/:id', (req, res) => {
  let data = fs.readFileSync(arquivo);
  let jogos = JSON.parse(data);
  let novoValor = req.body;

  let jogo = jogos.find(jogo => {
    if (jogo.id == req.params.id) {
      jogo.nome = novoValor.nome;
      jogo.categoria = novoValor.categoria;
      jogo.ano = novoValor.ano;
      fs.writeFileSync(arquivo, JSON.stringify(jogos));
      return jogo;
    }
  });

  if (jogo) {
    res.send(jogo);
  } else {
    res.status(404).send('Jogo não encontrado.');
  }
});
```

4 – Para finalizar, vamos criar um endpoint para deletar o registro usando o verbo DELETE.

```
app.delete('/jogos/:id', (req, res) => {
  let data = fs.readFileSync(arquivo);
  let jogos = JSON.parse(data);

  // Verifica se algum jogo foi removido
  if (!jogos.find(jogo => jogo.id == req.params.id)) {
    return res.status(404).send('Jogo não encontrado.');
  }

  // Filtra o array para remover o jogo com o id especificado
  let jogosAtualizados = jogos.filter(jogo => jogo.id != req.params.id);

  // Escreve o array atualizado de volta no arquivo
  fs.writeFileSync(arquivo, JSON.stringify(jogosAtualizados));
  res.send('Jogo removido com sucesso.');
});
```