

Objetivos:

- Estrutura de uma aplicação do lado servidor;
- Node e Express;
- Criar um servidor Node usando Express;
- Definir rotas;
- Servir arquivos estáticos.

Instruções: Antes de começar você precisa ter instalado o Node.js e NPM (Node Package Manager), use os comandos da Figura 1 para checar as versões instaladas.

Use o Visual Studio Code para criar a aplicação de teste.

```
Prompt de Comando

C:\>node -v
v12.13.1

C:\>npm -v
6.13.0

C:\>npx -v
6.13.0
```

Figura 1 – Comandos para obter a versão do Node.js e npm.

i. Estrutura de uma aplicação do lado servidor

Navegadores se comunicam com o servidor Web usando o protocolo HTTP. O Apache Tomcat, Apache PHP e Node Express são exemplos de servidores Web. A Figura 2 representa uma requisição HTTP, ela possui os objetos **Request** e **Response**.

O objeto **Request** inclui a URL (Uniform Resource Locator), o método que define a ação da requisição (GET, POST, PUT e DELETE), informações adicionais da URL, assim como os parâmetros e o corpo da requisição.

Na URL `http://localhost:3000?nome=Ana&idade=21`, **nome** e **idade** são parâmetros.

O objeto **Response** possui a mensagem de resposta com o status (200 OK, 404 Not Found etc.) e o corpo da resposta em caso de sucesso.

A representação da Figura 2 é de uma aplicação dinâmica, isto é, o **response** da requisição será de acordo com o resultado gerado pelo programa (Web Application). Existem também as aplicações estáticas, mas elas retornam sempre o mesmo conteúdo, isto é, elas não possuem a parte da Web Application.

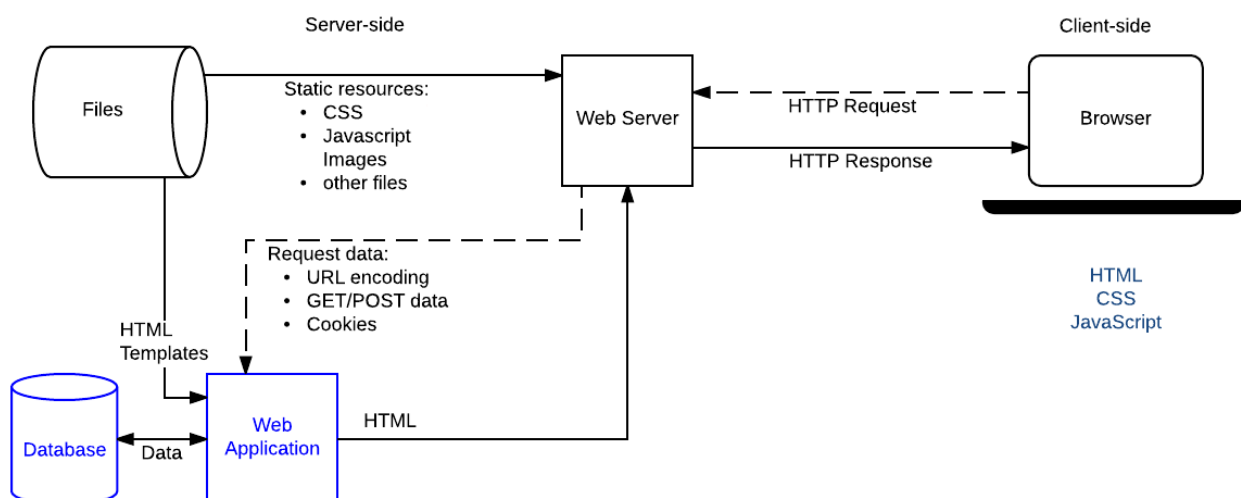


Figura 2 – Representação de uma requisição HTTP.

(Fonte: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Introduction)

ii. Node e Express

Node (ou formalmente Node.js) é um ambiente em tempo de execução open-source (código aberto) e multiplataforma que permite o desenvolvimento de aplicativos do lado servidor em JavaScript. Node destina-se a ser usado fora do contexto de um navegador, ou seja, executando diretamente no computador ou servidor. Como tal, o ambiente omite APIs JavaScript específicas do navegador e adiciona suporte para APIs de SO, incluindo bibliotecas de HTTP e manipulação de arquivos (https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction).

Podemos criar um servidor web usando apenas o pacote HTTP padrão do Node. Porém algumas tarefas do desenvolvimento web não são suportadas diretamente pelo próprio Node. Por exemplo, manipulação específica de requisições HTTP (GET, POST, PUT e DELETE), criação de caminhos de URL (rotas), servir arquivos estáticos ou usar modelos para criar dinamicamente a resposta. A solução é escrevermos esse código ou usarmos alguma biblioteca pronta que pode ser instalada usando NPM.

O Express é o framework web mais popular para Node, e é a biblioteca subjacente para uma série de outros frameworks.

iii. Criar um servidor Node usando Express

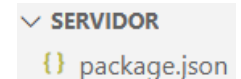
Use o VS Code para fazer a codificação. A seguir tem-se os passos para criar um servidor usando o framework Express (<https://expressjs.com/en/4x/api.html>).

- Crie uma pasta de nome **servidor** (pode ser qualquer outro nome de pasta) no local de sua preferência do computador;
- Abra a pasta **servidor** no VS Code;
- Acesse o terminal do VS Code e digite o comando **npm init -y**. Esse comando criará o arquivo **package.json**. Ao lado tem-se a estrutura de arquivos da pasta **servidor**;
- Digite **npm i express** para adicionar o pacote express. Como o projeto Node ainda não tinha sido criado, então ao instalar o pacote express foram adicionados os arquivos da biblioteca do Node na pasta **node_modules**. Ao lado tem-se a estrutura de arquivos da pasta **servidor**;
- Para melhor estruturar os arquivos. Crie a pasta **src** na raiz do projeto e, para começar, crie o arquivo **ex1.js** na pasta **src**. A seguir tem-se o conteúdo do arquivo **ex1.js**:

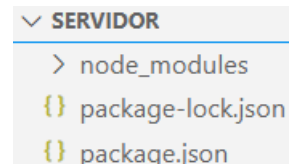
```
console.log("testando");
```

- Para executar um programa JavaScript usamos o comando **node** seguido pelo nome do arquivo. Como exemplo, digite o comando a seguir no terminal do VS Code:

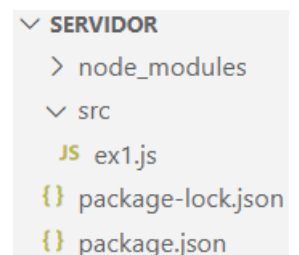
```
PS D:\aulas\servidor> node src/ex1.js  
testando
```



▼ SERVIDOR
{} package.json

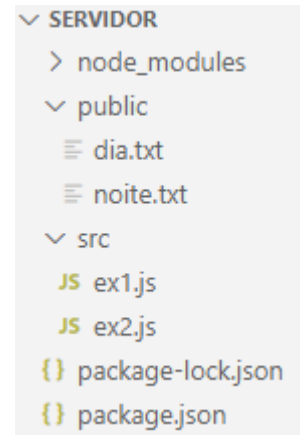


▼ SERVIDOR
> node_modules
{} package-lock.json
{} package.json



▼ SERVIDOR
> node_modules
▼ src
JS ex1.js
{} package-lock.json
{} package.json

- g) Crie também o arquivo `ex2.js`, a pasta `public` e os arquivos `dia.txt` e `noite.txt`. Assim como é mostrado ao lado. Dentro do arquivo `dia.txt` coloque apenas uma frase, por exemplo, **Bom dia!** e no arquivo `noite.txt` coloque **Boa noite!**.



- h) O comando `node` executa o arquivo uma única vez, já o comando `nodemon` executa o arquivo a cada modificação. O `nodemon` é indicado em tempo de desenvolvimento e o comando `node` em tempo de produção. Digite o comando `npm i nodemon --save-dev` para adicionar o pacote `nodemon` como dependência de desenvolvimento, ou seja, essa dependência não será incluída no deploy do projeto. Esse comando adicionará a propriedade `devDependencies` no arquivo `package.json`, observe que o pacote `express` está como dependência:

```
{
  "name": "servidor",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.12"
  }
}
```

Como exemplo digite o comando a seguir no terminal do VS Code para testar:

```
PS D:\aulas\servidor> nodemon src/ex1.js
[nodemon] 2.0.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/ex1.js`
testando
[nodemon] clean exit - waiting for changes before restart
```

Observação: em alguns poucos computadores o sistema operacional bloqueia o uso de scripts causando o seguinte erro ao executar o comando `nodemon`:

```
PS C:\Atividade\src> nodemon ex1.js
nodemon : O arquivo C:\Users\AppData\Roaming\npm\nodemon.ps1 não pode ser carregado porque a execução de scripts foi desabilitada
neste sistema. Para obter mais informações, consulte about_Execution_Policies em https://go.microsoft.com/fwlink/?LinkID=135170.
No linha:1 caractere:1
+ nodemon ex1.js
+ ~~~~~
+ CategoryInfo          : Erro de Segurança: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
```

Para resolver esse problema acesse <https://social.technet.microsoft.com/wiki/pt-br/contents/articles/35641.windows-10-permitir-a-execucao-de-scripts-no-powershell.aspx>.

- i) Coloque o código a seguir no arquivo `src/ex1.js`. O serviço estará na porta 3001.

```
const express = require('express'); //importa o pacote Express
const app = express(); //cria uma aplicação express
// para receber parâmetros enviados pelo corpo da requisição
app.use(express.urlencoded({ extended: true }));

// define a porta e a função callback a ser executada após o servidor iniciar
app.listen(3001, function () {
    console.log("Servidor rodando na porta 3001...");
});

// definição da rota para a raiz usando o método HTTP GET
// curl -X GET http://localhost:3001/
app.get('/', function (req, res) {
    // o objeto que está na variável res possui os dados a serem enviados para o cliente
    res.send('Raiz');
});

// curl -X GET http://localhost:3001/cadastro/cliente
app.get('/cadastro/cliente', function (req, res) {
    res.send('caminho para cadastro/cliente');
});

// essa rota recebe 2 parâmetros pela própria URL
// curl -X GET http://localhost:3001/produto/arroz/10.99
app.get('/produto/:nome/:valor', function (req, res) {
    // O objeto que está na variável req possui tudo
    // que foi enviado pelo cliente na requisição
    // Os valores passados como parâmetro estão na propriedade params
    let nome = req.params.nome;
    let valor = req.params.valor;
    res.send('parâmetros ' + nome + ' e ' + valor);
});

// curl -X GET -d "x=5&y=2" http://localhost:3001/valores
app.get('/valores', function (req, res) {
    // recebe os parâmetros enviados pelo corpo da requisição
    let { x, y } = req.body;
    res.send('GET ' + x + ' e ' + y);
});
```

```
});

// definição da rota para a raiz usando o método HTTP POST
// curl -X POST -d "x=5&y=2" http://localhost:3001/valores
app.post('/valores', function (req, res) {
  //recebe os parâmetros enviados pelo corpo da requisição
  let { x, y } = req.body;
  res.send('POST ' + x + ' e ' + y);
});

// curl -X POST -d "x=5&y=2" http://localhost:3001/valores/10
app.post('/valores/:w', function (req, res) {
  let { x, y } = req.body; //recebe os parâmetros enviados pelo corpo da requisição
  let w = req.params.w; //parâmetro enviado pela URL
  res.send('POST ' + x + ', ' + y + ' e ' + w);
});

// o método all processa requisições oriundas de qualquer método HTTP
//curl -X GET -d "x=1&y=2" http://localhost:3001/tudo
//curl -X POST -d "x=3&y=4" http://localhost:3001/tudo
//curl -X PUT -d "x=5&y=6" http://localhost:3001/tudo
//curl -X DELETE -d "x=7&y=8" http://localhost:3001/tudo
app.all('/tudo', function (req, res){
  let { x, y } = req.body;
  res.send('ALL ' + x + ' e ' + y);
});

//aceita qualquer método HTTP ou URL iniciando por /inicio
//curl -X POST http://localhost:3001/inicio/teste
app.use('/inicio', function(req, res){
  res.send('URL com /inicio');
});

//aceita qualquer método HTTP e URL
app.use(function(req, res){
  res.send('URL desconhecida');
});
```

O método `listen` (<https://expressjs.com/en/4x/api.html#app.listen>) é usado para iniciar o servidor na porta indicada. A função callback passada como 2º parâmetro é opcional e será invocada logo após o servidor ser iniciado.

```
//define a porta e a função callback a ser executada após o servidor ser iniciado
app.listen(3001, function () {
  console.log("Servidor rodando na porta 3001...");
});
```

j) Coloque o código a seguir no arquivo `src/ex2.js`, esse código é usado para entregar arquivos estáticos na porta 3002.

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

```
app.listen(3002, function () {
  console.log("Servidor rodando na porta 3002...");
});

// rota para a pasta public
// http://localhost:3002/saudacao/dia.txt
// http://localhost:3002/saudacao/noite.txt
app.use('/saudacao', express.static('public'));

// rota para um arquivo específico
// http://localhost:3002/teste
app.use('/teste', express.static('public/noite.txt'));
```

iv. Definir rotas

O Express possui métodos para manipular rotas de requisições HTTP GET (app.get()), HTTP POST (app.post()), HTTP PUT (app.put()) e HTTP DELETE (app.delete()). Podemos definir quantas rotas quisermos, porém não podem existir duas rotas mapeadas para a mesma URL + método HTTP.

Importante: não poderemos acessar o servidor se nenhuma rota for definida.

No exemplo foram definidas as seguintes rotas:

1 - Rota para a URL <http://localhost:3001/> usando o método HTTP GET. Observe que estamos roteando para a raiz.

```
app.get('/', function(req, res) {
  res.send('Raiz');
});
```

O 1º parâmetro do método `get` define o caminho, neste caso foi a raiz `/`, e o 2º parâmetro define a função callback.

As requisições HTTP possuem os objetos `request` (com os dados enviados pelo cliente) e `response` (com os dados a serem enviados pelo servidor para o cliente). Esses objetos são passados para a função callback nos parâmetros `req` e `res`, desta forma, usamos o método `send` do objeto `Request` (<https://expressjs.com/en/4x/api.html#req>) para enviar o texto `'Raiz'` para o cliente.

2 - Rota para a URL <http://localhost:3001/cadastro/cliente> usando o método HTTP GET. Observe que criamos uma estrutura como se fossem pastas `/cadastro/cliente`, mas na prática elas não existem.

```
app.get('/cadastro/cliente', function (req, res) {
  res.send('caminho para cadastro/cliente');
});
```

3 - Rota para a URL <http://localhost:3001/produto/arroz/10.99> usando o método HTTP GET. Os parâmetros da URL são definidos usando dois pontos seguido do nome do parâmetro. A ordem dos parâmetros precisa ser seguida na chamada da URL, neste exemplo não podemos trocar arroz por 10.99. Para ler os parâmetros na função callback precisamos acessar o objeto `req` (Request).

```
app.get('/produto/:nome/:valor', function (req, res) {
  let nome = req.params.nome;
  let valor = req.params.valor;
```

```
res.send('parâmetros ' + nome + ' ' + valor);
});
```

4 - Rota para a URL <http://localhost:3001/valores> usando o método HTTP GET. A propriedade `req.body` (<https://expressjs.com/en/4x/api.html#req.body>) contém os dados submetidos no corpo da requisição no formato key-value.

```
app.get('/valores', function (req, res) {
  //recebe os parâmetros enviados pelo corpo da requisição
  let { x, y } = req.body;
  res.send('GET ' + x + ' e ' + y);
});
```

Como não temos uma aplicação cliente, então, para testar, teremos de usar o comando CURL no prompt do DOS

```
curl -X GET -d "x=5&y=2" http://localhost:3001/valores
```

No comando CURL usamos o parâmetro `-d` seguido pelos valores passados através do corpo e `-X` para indicar o método HTTP da requisição.

5 - Rota para a URL <http://localhost:3001/valores> usando o método HTTP POST. Como a URL é igual a anterior, então o servidor as diferencia pelo método HTTP.

```
app.post('/valores', function (req, res) {
  //recebe os parâmetros enviados pelo corpo da requisição
  let { x, y } = req.body;
  res.send('POST ' + x + ' e ' + y);
});
```

Para testar teremos de informar o método HTTP POST no comando CURL

```
curl -X POST -d "x=5&y=2" http://localhost:3001/valores
```

6 - Rota para a URL <http://localhost:3001/valores/10> usando o método HTTP POST. Podemos passar parâmetros na URL e no corpo da requisição. No exemplo a seguir o parâmetro `w` é passado pela URL e os parâmetros `x` e `y` pelo corpo da requisição.

```
app.post('/valores/:w', function (req, res) {
  //recebe os parâmetros enviados pelo corpo da requisição
  let { x, y } = req.body;
  let w = req.params.w;
  res.send('POST ' + x + ', ' + y + ' e ' + w);
});
```

Para testar teremos de informar o método HTTP POST no comando CURL

```
curl -X POST -d "x=5&y=2" http://localhost:3001/valores/10
```

7 – O método `all` define uma rota que é invocada para qualquer método HTTP.

```
app.all('/tudo', function (req, res){
  let { x, y } = req.body;
  res.send('ALL ' + x + ' e ' + y);
});
```

Para testar podemos fornecer qualquer um dos seguintes comandos, veja que cada um deles utiliza um método HTTP diferente:

```
curl -X GET -d "x=1&y=2" http://localhost:3001/tudo
curl -X POST -d "x=3&y=4" http://localhost:3001/tudo
curl -X PUT -d "x=5&y=6" http://localhost:3001/tudo
curl -X DELETE -d "x=7&y=8" http://localhost:3001/tudo
```

8 – O método `use` define um caminho padrão. Desta forma qualquer caminho que se inicia pela rota padrão será aceito, também aceita todos os tipos de método HTTP. No exemplo a seguir qualquer método HTTP e URL iniciando por `/inicio` será aceita, pois `/inicio` é o caminho padrão.

```
app.use('/inicio', function(req, res){
  res.send('URL com /inicio');
});
```

Para testar podemos fornecer qualquer um dos seguintes comandos

```
curl -X GET -d "x=1&y=2" http://localhost:3001/inicio
curl -X POST -d "x=3&y=4" http://localhost:3001/inicio/abc/2
curl -X PUT -d "x=5&y=6" http://localhost:3001/inicio/val
curl -X DELETE -d "x=7&y=8" http://localhost:3001/inicio/val
```

9 – Podemos usar o método `use` sem o 1º parâmetro para definir uma rota genérica, isto é, qualquer caminho é aceito.

```
app.use(function(req, res){
  res.send('URL desconhecida');
});
```

Para testar podemos fornecer qualquer um dos seguintes comandos. Veja que elas são rotas não mapeadas pelo servidor, porém elas serão capturadas pela rota aqui definida.

```
curl -X GET -d "x=1&y=2" http://localhost:3001/x
curl -X POST -d "x=3&y=4" http://localhost:3001/inicios
curl -X PUT -d "x=5&y=6" http://localhost:3001/teste
curl -X DELETE -d "x=7&y=8" http://localhost:3001/val/2
```

Importante: a ordem de definição das rotas é muito importante, se colocarmos essa definição de rota como a 1ª, então nenhuma outra rota será avaliada. As rotas mais generalistas devem estar no final.

Aqui usamos o método `send` para enviar a resposta para o cliente, existem outras opções, assim como o método `json` (<https://expressjs.com/en/4x/api.html#res.json>). De qualquer forma é necessário enviar uma resposta para a solicitação não ficar pendente no cliente e não terminar.

v. Servir arquivos estáticos

A middleware `express.static` (<http://expressjs.com/en/4x/api.html#express.static>) é usada para servir arquivos estáticos (HTML, CSS, PNG, TXT etc.), isto é, arquivos que são enviados para o cliente assim como eles se encontram.

Podemos criar um prefixo virtual para as URLs estáticas. O 1º parâmetro do método `use` define o prefixo a ser adicionado na URL e o segundo parâmetro é o mapeamento para o arquivo ou pasta. No exemplo a seguir a pasta `public` foi mapeada para a rota `/saudacao`:


```
app.use('/saudacao', express.static('public'));
```

Para acessar o arquivo da pasta `/public/dia.txt` basta fornecer a URL

```
http://localhost:3002/saudacao/dia.txt
```

O exemplo a seguir mapeia um arquivo para a rota `/teste`:

```
app.use('/teste', express.static('public/noite.txt'));
```

A URL será para a raiz se for omitido o 1º parâmetro. Como exemplo, a rota `app.use(express.static('public'))` seria acessada usando a URL `http://localhost:3002/dia.txt`.