Objetivos:

I. Estrutura de uma aplicação do lado servidor;
II. Node.js e Express;
III. Criar servidor Node.js com Express;
IV. Definição de rotas;
V. Envio de dados para o servidor.

I. Estrutura de uma aplicação do lado servidor

Os navegadores se comunicam com servidores web usando o protocolo HTTP (Hypertext Transfer Protocol). Servidores como Apache Tomcat, Apache com PHP e Node.js com Express são exemplos comuns para rodar aplicações no back-end.

A Figura 1 representa uma requisição HTTP, que envolve dois principais objetos:

- Request (Requisição): contém informações enviadas pelo cliente, como:
  - URL (Uniform Resource Locator): define o endereço do recurso solicitado;
  - Método HTTP: define a ação a ser realizada (ex.: GET, POST, PUT, DELETE);
  - Parâmetros e corpo da requisição: incluem dados adicionais necessários para processar a solicitação.

  Exemplo de URL com parâmetros:

  `http://localhost:3000?nome=Ana&idade=21`

  nome e idade são parâmetros passados para o servidor.

  3000 é a porta no servidor.
- Response (Resposta): contém a mensagem de retorno do servidor, que inclui:
  - Código de status HTTP: indica o resultado da requisição (ex.: 200 OK, 401 Unauthorized, 404 Not Found);
  - Corpo da resposta: pode conter dados, mensagens ou arquivos.

Aplicações dinâmicas x estáticas:

- Aplicações dinâmicas: o conteúdo do objeto Response é gerado de acordo com o resultado processado pelo programa (ex.: consultas a um BD, cálculos ou lógica específica);
- Aplicações estáticas: sempre retornam o mesmo conteúdo, geralmente arquivos fixos, como HTML, CSS ou imagens.

Códigos de status HTTP: as respostas HTTP utilizam códigos de status de três dígitos para informar o resultado de uma requisição. Eles são divididos em cinco classes principais:

- 100 a 199 (respostas de informação): indica que a requisição foi recebida e o servidor continua processando-a;

- 200 a 299 (respostas de sucesso): indica que a requisição foi recebida, entendida e aceita com sucesso:
  - 200 OK: a requisição foi bem-sucedida;
  - 201 Created: a requisição foi bem-sucedida e resultou na criação de um novo recurso;
  - 204 No Content: a requisição foi bem-sucedida, mas não há conteúdo para ser retornado (por exemplo, em uma requisição DELETE).
- 300 a 399 (redirecionamentos): indica que a requisição precisa de ações adicionais para ser concluída:
  - 301 Moved Permanently: a URI do recurso solicitado foi alterada permanentemente e a nova URI é fornecida na resposta;
  - 302 Found / 303 See Other: a URI do recurso solicitado foi temporariamente alterada. O cliente deve redirecionar para a URI fornecida na resposta;
  - 304 Not Modified: indica que o recurso solicitado não foi modificado desde a última requisição.
- 400 a 499 (erros do cliente): indica que houve um erro por parte do cliente na requisição:
  - 400 Bad Request: a requisição foi malformada ou incompreensível para o servidor;
  - 401 Unauthorized: o cliente não foi autorizado a acessar o recurso;
  - 403 Forbidden: o cliente não tem permissão para acessar o recurso;
  - 404 Not Found: o recurso solicitado não foi encontrado no servidor;
  - 409 Conflict: o servidor não pode completar a requisição devido a um conflito no estado atual do recurso.
- 500 a 599 (erros do servidor): indica que houve um erro no servidor ao processar a requisição:
  - 500 Internal Server Error: o servidor encontrou uma situação inesperada que o impediu de atender à requisição;
  - 502 Bad Gateway: o servidor atuando como um gateway ou proxy recebeu uma resposta inválida do servidor upstream;
  - 503 Service Unavailable: o servidor não está pronto para lidar com a requisição. Geralmente, isso ocorre quando o servidor está em manutenção ou sobrecarregado.

Esses são alguns dos códigos de erro mais comuns usados em respostas de requisição HTTP. Compreender esses códigos é essencial para interpretar corretamente as respostas do servidor e lidar com diferentes cenários durante a comunicação entre cliente e servidor. Para mais detalhes https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status.

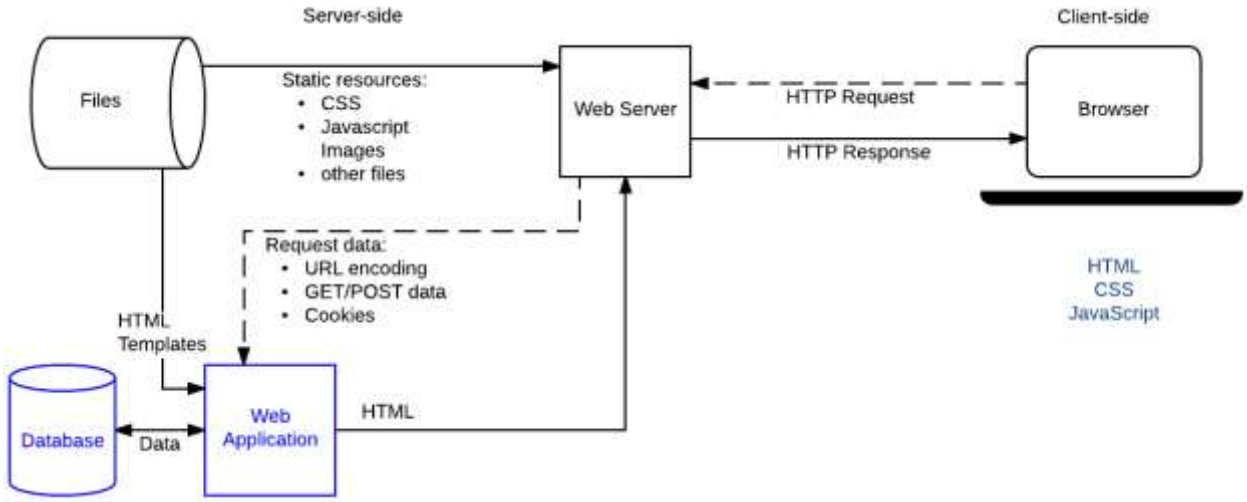


Figura 1 – Representação de uma requisição HTTP.

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

II. Node.js e Express

Node.js é um ambiente de execução (*runtime*) open-source e multiplataforma que permite a execução de código JS no lado do servidor. Ele foi projetado para ser usado fora do contexto de um navegador, funcionando diretamente em um computador ou servidor. Dessa forma, o Node.js omite APIs específicas do navegador (como DOM e window) e adiciona suporte para APIs do sistema operacional, incluindo bibliotecas HTTP e manipulação de arquivos. Para mais detalhes https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction.

É possível criar um servidor web usando apenas o módulo HTTP padrão do Node.js. Contudo, tarefas comuns no desenvolvimento web não são diretamente suportadas pelo módulo nativo, como:

- Manipulação detalhada de requisições HTTP (GET, POST, PUT, DELETE);
- Criação de rotas dinâmicas e específicas para URLs;
- Servir arquivos estáticos (imagens, CSS, JS);
- Uso de modelos (templates) para geração dinâmica de conteúdo.

Para resolver essas limitações, os desenvolvedores têm duas opções:

- Escrever manualmente todo o código necessário para essas funcionalidades;
- Utilizar bibliotecas ou frameworks especializados, como o Express.

O Express é um framework minimalista e amplamente utilizado para desenvolvimento de aplicações web e APIs com Node.js. Ele fornece uma abstração de alto nível para simplificar tarefas comuns, como:

- Roteamento: definição de caminhos e regras para URLs específicas;
- Tratamento de requisições e respostas HTTP: processamento de dados recebidos e envio de respostas adequadas;

- Middlewares: funções intermediárias que modificam requisições ou respostas antes de chegarem ao destino;
- Servir arquivos estáticos: disponibilizar arquivos públicos como CSS, JS e imagens;
- Suporte para diferentes formatos de resposta: JSON, HTML, texto simples, entre outros.

III. Criar servidor Node.js com Express

A seguir, tem-se os passos para configurar um servidor Node.js utilizando o framework Express (https://www.npmjs.com/package/express):

1. Criar uma pasta para o projeto
   - Crie uma pasta no seu computador. Neste exemplo, ela se chamará server, mas pode ser qualquer outro nome, desde que não contenha caracteres especiais ou espaços;
2. Inicializar o projeto Node.js
   - Abra a pasta no terminal (CMD ou terminal integrado do VS Code);
   - Execute o comando `npm init -y`. O parâmetro `-y` evita perguntas interativas e cria automaticamente o arquivo package.json com configurações padrão;

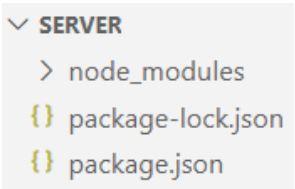
```
C:\Windows\System32\cmd.e
D:\pessoal\server>npm init -y
Wrote to D:\pessoal\server\package.json:
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}
```

3. Instalar o Express

   ```
   npm i express
   ```

   Esse comando cria a pasta `node_modules` e adiciona a dependência no `package.json`.

   Ao final desse passo o projeto terá a estrutura mostrada ao lado.



4. Criar a estrutura do projeto (assim como é mostrado ao lado)

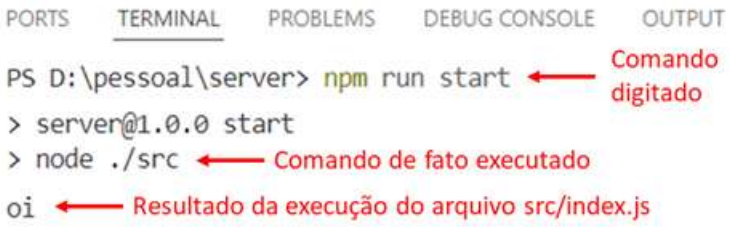
   - No VS Code, crie uma pasta chamada `src` para armazenar os arquivos JS. Constitui boa prática manter o código na pasta `src`;
   - Dentro da pasta `src`, crie um arquivo chamado `index.js`;
   - Adicione a seguinte instrução no arquivo `index.js`:

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

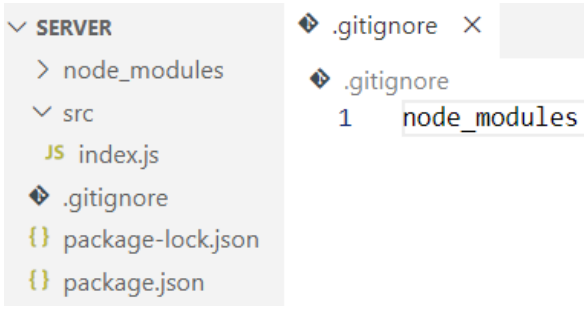
5. Adicionar scripts no `package.json`
   - No arquivo `package.json`, adicione a propriedade `start` em `scripts` para executarmos o projeto:

```
"scripts": {
  "start": "node ./src"
},
```

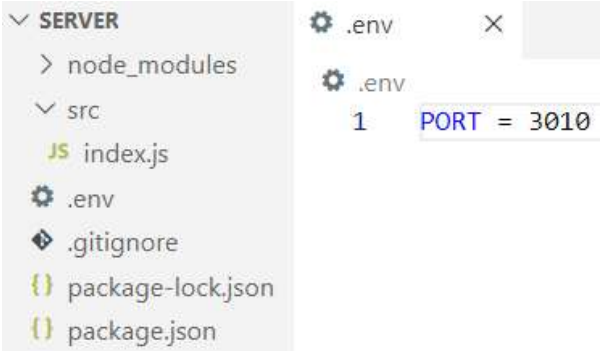
   - Execute o projeto no terminal do VS Code:



6. Crie o arquivo `.gitignore` no raiz do projeto e coloque a instrução `node_modules`.
   Isso evitará que a pasta `node_modules` seja enviada para o repositório do GitHub.



7. Crie o arquivo .env no raiz do projeto e coloque a instrução `PORT = 3010` (não pode ter ponto e vírgula).



O arquivo .env é usado no projeto Node para armazenar variáveis de ambiente. Essas

variáveis são informações sensíveis ou configurações específicas do ambiente que o aplicativo precisa para funcionar corretamente. Elas podem incluir senhas, chaves de API, credenciais de BD etc.

A função do arquivo .env é fornecer uma maneira de definir e gerenciar variáveis de ambiente sem incluí-las diretamente no código fonte. Desta forma, através do .gitignore podemos excluir o arquivo .env do controle de versões, evitando a exposição de dados sensíveis. Cada desenvolvedor deve criar seu próprio arquivo .env com suas próprias configurações de ambiente específicas.

8. As variáveis do arquivo .env não são carregadas pelo ambiente de execução do Node. Para resolver esse problema temos de instalar o pacote dotenv (https://www.npmjs.com/package/dotenv). Digite o seguinte comando no terminal do VS Code:

```
npm i dotenv
```

Observe que os nomes e versões do pacote dotenv são incluídos nas propriedades dependencies do arquivo package.json:

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node ./src"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "dotenv": "^16.4.7",
    "express": "^4.21.2"
  }
}
```

A numeração de versões em um pacote segue geralmente o formato "major.minor.patch", conforme o padrão SemVer (Semantic Versioning):

- major (principal): alterações que quebram compatibilidade com versões anteriores;
- minor (secundária): adição de novos recursos que não afetam a compatibilidade existente;
- patch (correção): correções de bugs ou melhorias que não afetam funcionalidades

existentes.

Significado do símbolo ^ no package.json: permite que o gerenciador de pacotes (npm ou yarn) atualize automaticamente as versões minor e patch, mas não altera a versão major. Exemplo: `"express": "^4.21.2"` permite instalar versões como:

- `4.21.3` (patch atualizado);
- `4.22.0` (minor atualizado);
- No entanto, não permitirá instalar uma versão `5.0.0`, pois isso representaria uma mudança na versão major e, potencialmente, uma quebra de compatibilidade.

9. Adicione o seguinte código no arquivo `src/index.js` para testar o uso da variável PORT declarada no arquivo .env:

```
PORTS  TERMINAL  PROBLEMS  DEBUG CONSOLE  OUTPUT
PS D:\pessoal\server> npm run start
> server@1.0.0 start
> node ./src
Servidor rodando na porta 3010
```

```
// Importa a biblioteca dotenv para carregar variáveis de ambiente do arquivo .env
const dotenv = require('dotenv');

// Carrega as variáveis de ambiente do arquivo .env para o process.env
dotenv.config();

// Obtém a variável de ambiente PORT definida no arquivo .env ou usa 3000 como padrão
const port = process.env.PORT || 3000;

// Exibe no console a porta em que o servidor será executado
console.log(`Servidor rodando na porta ${port}`);
```

IV. Definição de rotas

Uma rota define um caminho específico no servidor para o qual as solicitações dos clientes são direcionadas, bem como a forma como essas solicitações serão tratadas. Clientes comuns incluem navegadores, aplicativos móveis ou outros sistemas que fazem requisições HTTP.

Estrutura de uma rota: uma rota é formada pela combinação de dois elementos principais:

- URL: é o caminho que os clientes usam para acessar um recurso específico no servidor. Geralmente começa com o domínio (ou IP) do servidor, seguido por uma porta e por um caminho que aponta para um endpoint específico.
  - Exemplo: `http://localhost:3010/teste` representa um recurso acessado pelo caminho `/teste`.
- Método HTTP: define a ação que o cliente deseja realizar no recurso especificado pela URL. Alguns dos métodos mais comuns são:

- GET: usado para solicitar dados do servidor. Geralmente associado a operações de leitura, como um SELECT em um BD;
- POST: usado para enviar dados ao servidor para serem processados, como um INSERT em umBD;
- PUT: usado para atualizar dados um recurso existente, como um UPDATE em um BD;
- DELETE: usado para remover dados de um recurso específico, com um DELETE em um BD.

Outros métodos HTTP menos comuns incluem HEAD, PATCH, OPTIONS, entre outros.

No Express, há métodos correspondentes para cada tipo de método HTTP: get, post, put, delete, entre outros. Coloque o código a seguir no arquivo `src/index.js`:

```
// Importa o framework Express para criar e gerenciar o servidor web
const express = require("express");
const dotenv = require("dotenv");

dotenv.config();

// Cria uma instância do aplicativo Express
const app = express();

const port = process.env.PORT || 3000;

// Inicia o servidor na porta definida e exibe uma mensagem no console
app.listen(port, function () {
  console.log(`Servidor rodando na porta ${port}`);
});

// Rota GET para a raiz "/" que retorna uma mensagem ao usuário
app.get("/", function(req, res) {
  res.send("Método HTTP GET");
});

// Rota GET para "/teste" que retorna uma mensagem ao usuário
app.get("/teste", function(req, res) {
  res.send("Método HTTP GET para /teste");
});

// Rota POST para "/teste" que retorna uma mensagem ao usuário
app.post("/teste", function(req, res) {
  res.send("Método HTTP POST para /teste ");
});

// Rota PUT para "/teste" que retorna uma mensagem ao usuário
app.put("/teste", function(req, res) {
  res.send("Método HTTP PUT para /teste ");
});

// Rota DELETE para "/teste" que retorna uma mensagem ao usuário
```

```
app.delete("/teste", function(req, res) {
  res.send("Método HTTP DELETE para /teste ");
});
```

O método `listen` do Express é responsável por iniciar o servidor e fazê-lo escutar requisições em uma porta específica. Ele cria um servidor HTTP e permite que a aplicação processe solicitações enviadas pelos clientes, como navegadores ou outras aplicações. Sintaxe básica:

```
app.listen(port, callback);
```

`port`: número da porta na qual o servidor será executado. Pode ser um valor fixo ou obtido de uma variável de ambiente (`process.env.PORT`).

`callback` (opcional): função executada quando o servidor inicia com sucesso. Normalmente, essa função exibe uma mensagem no console informando que o servidor está em execução.

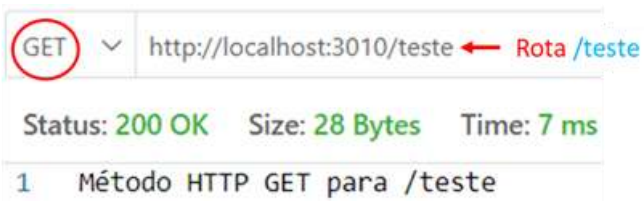
Ao executar o projeto o terminal ficará travado. Use as teclas <Ctrl> + <C> para cancelar a execução e liberar o terminal para receber novos comandos.

> Observação: no navegador você consegue testar apenas as rotas que usam o método HTTP GET. Para testar as demais rotas sugere-se instalar as extensões **Thunder Client** (paga) ou **EchoAPI for VS Code** (gratuita) no VS Code.

Teste das rotas:

Requisição HTTP GET para o endpoint /

Requisição HTTP GET para o endpoint /teste



Requisição HTTP POST para o endpoint /teste

Requisição HTTP PUT para o endpoint /teste

O curl (Client URL) é uma ferramenta de linha de comando usada para realizar requisições HTTP. Ele permite enviar e receber dados de servidores web diretamente pelo terminal (prompt de comando Windows).

Estrutura de um comando curl:

```
curl [opções] [URL] [dados]
```

- `curl`: indica o uso do comando.
- [opções]: define o método HTTP, cabeçalhos, autenticação, entre outros parâmetros.
- [URL]: especifica o endereço do servidor;
- [dados]: no caso de requisições POST, PUT ou PATCH, permite enviar dados no corpo da requisição.

Exemplos:

- Requisição HTTP GET para o endpoint /teste

```
curl -X GET http://localhost:3010/teste
```

- Requisição HTTP POST para o endpoint /teste

```
curl -X POST http://localhost:3010/teste
```

```
D:\>curl -X POST http://localhost:3010/teste
Método HTTP POST para /teste
```

- Requisição HTTP PUT para o endpoint /teste

```
curl -X PUT http://localhost:3010/teste
```

```
D:\>curl -X PUT http://localhost:3010/teste
Método HTTP PUT para /teste
```

- Requisição HTTP DELETE para o endpoint /teste

```
curl -X DELETE http://localhost:3010/teste
```

```
D:\>curl -X DELETE http://localhost:3010/teste
Método HTTP DELETE para /teste
```

V. Envio de dados para o servidor

Para que um cliente possa interagir com o servidor de forma eficiente, é necessário enviar dados de diferentes maneiras, dependendo do propósito e do tipo de requisição. Os principais métodos para envio de dados são:

1. Parâmetros na URL (Route Parameters);

2. Query Parameters;
3. Body (Corpo da Requisição).

Cada método tem uma aplicação específica e pode ser usado em diferentes cenários.

1. Parâmetros na URL

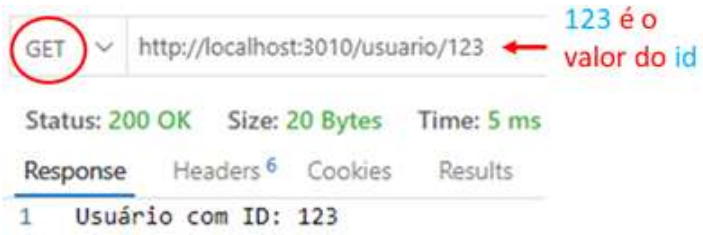
Os parâmetros na URL são usados para enviar informações diretamente no caminho da rota. Geralmente, são utilizados para identificar recursos específicos no servidor, como um ID ou um nome. Exemplo de uso:

```
// Rota com parâmetro dinâmico na URL
app.get("/usuario/:id", (req, res) => {
  const { id } = req.params; // Pegar o valor passado na URL
  res.send(`Usuário com ID: ${id}`);
});
```

Explicação:

- A URL `/usuario/:id` define um parâmetro dinâmico chamado `id`;
- O valor do parâmetro pode ser acessado através de `req.params.id`, ou por {desestruturação} do objeto `params`, assim como em `const { id } = req.params`.

Exemplo de requisição. O valor 123 será atribuído ao parâmetro `id`.



Teste no terminal do prompt de comando do Windows:

```
curl -X GET http://localhost:3010/usuario/123
```

```
D:\>curl -X GET http://localhost:3010/usuario/123
Usuário com ID: 123
```

2. Query Parameters

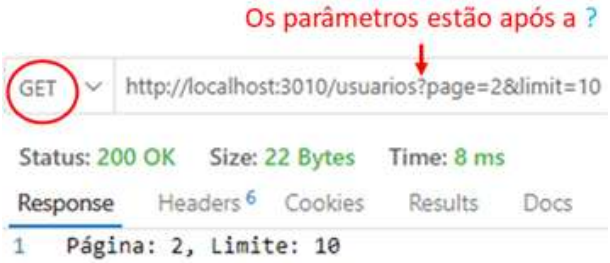
Os Query Parameters permitem enviar dados adicionais na URL por meio de pares `chave=valor`. Eles são frequentemente usados para filtragem, paginação ou parâmetros opcionais. Exemplo de uso:

```
// Rota com Query Parameters
app.get("/usuarios", (req, res) => {
  const { page, limit } = req.query; // Pegar os parâmetros passados na URL
  res.send(`Página: ${page}, Limite: ${limit}`);
});
```

Explicação:

- Os parâmetros são adicionados após o símbolo ? na URL;
- Parâmetros múltiplos são separados por &;
- Os valores podem ser acessados através de `req.query`.

Exemplo de requisições:



Teste no terminal do prompt de comando do Windows:

```
curl -X GET "http://localhost:3010/usuarios?page=2&limit=10"
```

```
D:\>curl -X GET "http://localhost:3010/usuarios?page=2&limit=10"
Página: 2, Limite: 10
```

Observação: o prompt interpreta o caractere `&` como um separador de comandos, tentando executar `limit=10` como um comando separado. A maneira mais simples de evitar esse problema é envolver a URL com aspas duplas.

3. Body (Corpo da Requisição)

O Body é usado para enviar dados estruturados no corpo da requisição, geralmente em formatos como JSON. Esse método é ideal para envio de informações mais complexas ou sensíveis, como formulários, dados de cadastro, ou arquivos. Exemplo de uso:

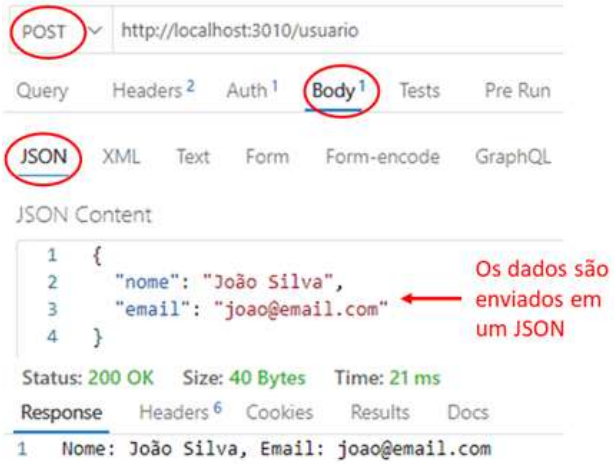
```
app.use(express.json());

// Rota com Body
app.post("/usuario", (req, res) => {
  const { nome, email } = req.body;
  res.send(`Nome: ${nome}, Email: ${email}`);
});
```

Explicação:

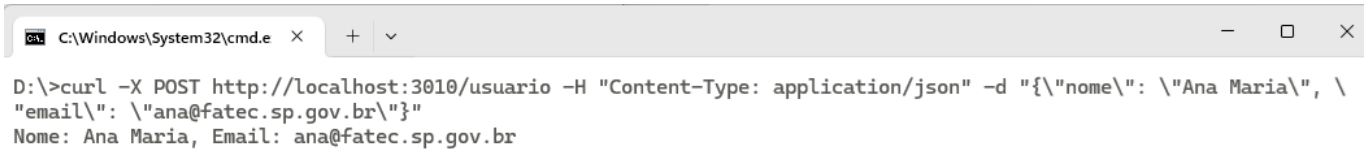
- O middleware `express.json()` é necessário para que o servidor consiga interpretar o corpo da requisição em formato JSON;
- Os dados são acessados através de `req.body`.

Exemplo de Requisição (usando JSON). É necessário usar o Thunder Client ou algum outro recurso capaz de fazer requisições do tipo POST:



Teste no terminal do prompt de comando do Windows:

```
curl -X POST http://localhost:3010/usuario -H "Content-Type: application/json" -d "{\"nome\": \"Ana Maria\", \"email\": \"ana@fatec.sp.gov.br\"}"
```



Observação: as aspas do objeto JSON precisam ser escapadas com \.

Exercícios

Veja os vídeos se tiver dúvidas nos exercícios:

Exercício 1 - https://youtu.be/sedjyLO3uzg

Exercícios 2 a 5 - https://youtu.be/4loSiRdyPBM

Exercício 1 – Complete o código a seguir para atender aos seguintes requisitos:

- O servidor deve responder na porta 3101, que deve ser configurada no arquivo .env;
- Crie uma rota utilizando o método HTTP GET que recebe um nome e um índice como parâmetros na URL, por exemplo: /texto/nome/indice; e retorna um objeto JSON no formato {letra:string}
  - O nome e o índice devem ser parâmetros na rota;
  - A rota deverá retornar a letra correspondente à posição indicada pelo índice no nome fornecido;
  - Exemplo, /texto/Maria/0 deve retornar M, pois a letra M está na posição 0 da string Maria.

```
curl -X GET http://localhost:3101/texto/Maria/0
```

```
D:\>curl -X GET http://localhost:3101/texto/Maria/0
{"letra":"M"}
```

Dicas:

- Os parâmetros são recebidos com string, use `parseInt` para converter para número;
- Use o método `json` do objeto que está na variável `res`. Por exemplo: `res.json({ letra: nome[i] });`

```
const express = require("express");
```

```
const dotenv = require("dotenv");

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;

app.use(express.json());

app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});
```

Exercício 2 – Adicione ao código do Exercício 1 uma rota que recebe o nome e índice utilizando Query Parameters em vez de parâmetros na URL.

Requisitos:

- A rota deverá utilizar o método HTTP GET;
- A rota deverá retornar um objeto JSON no formato {letra:string};
- Os parâmetros nome e indice devem ser recebidos via query string, por exemplo:

```
curl -X GET "http://localhost:3101/texto?nome=Maria&indice=0"
```

```
D:\>curl -X GET "http://localhost:3101/texto?nome=Maria&indice=0"
{"letra":"M"}
```

Exercício 3 – Adicione ao código do Exercício 2 uma rota que recebe o nome e índice pelo corpo da requisição (request body) no formato JSON.

Requisitos:

- A rota deverá utilizar o método HTTP POST;
- A rota deverá retornar um objeto JSON no formato {letra:string};
- Os parâmetros nome e indice devem ser enviados no corpo da requisição no formato JSON.

Dica:

- Certifique-se de que o servidor está configurado para processar o corpo da requisição em formato JSON, utilizando o middleware `express.json()`.

```
curl -X POST http://localhost:3101/texto -H "Content-Type: application/json" -d "{\"nome\": \"Maria\", \"indice\": \"0\"}"
```

```
D:\>curl -X POST http://localhost:3101/texto -H "Content-Type: application/json" -d "{\"nome\": \"Maria\",
 \"indice\": \"0\"}"
{"letra":"M"}
```

Exercício 4 – Adicione ao código do Exercício 3 uma rota que recebe dois números reais utilizando Query Parameters e retorna a soma deles.

Requisitos:

- A rota deverá utilizar o método HTTP GET;
- A rota deverá retornar um objeto JSON no formato {resultado:number}.

Dica:

- Os parâmetros são recebidos com string, use `parseFloat` para converter para número.

```
curl -X GET "http://localhost:3101/soma?x=8.2&y=9.3"
```

```
D:\>curl -X GET "http://localhost:3101/soma?x=8.2&y=9.3"
{"resultado":17.5}
```

Exercício 5 – Adicione ao código do Exercício 4 uma rota que recebe dois números reais pelo corpo da requisição (request body) no formato JSON e retorna à potência.

Requisitos:

- A rota deverá utilizar o método HTTP POST;
- A rota deverá retornar um objeto JSON no formato {resultado:number}.

Dica:

- Os parâmetros são recebidos com string, use `parseFloat` para converter para número.

```
curl -X POST http://localhost:3101/potencia/ -H "Content-Type: application/json" -d "{\"base\": \"2\", \"expoente\": \"3\"}"
```

```
D:\>curl -X POST http://localhost:3101/potencia/ -H "Content-Type:
application/json" -d "{\"base\": \"2\", \"expoente\": \"3\"}"
{"resultado":8}
```