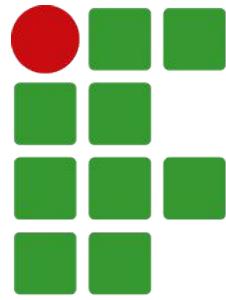


INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Protocolo HTTP Framework Express

Desenvolvimento Web

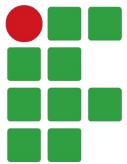
Prof. Diego Stiehl



INSTITUTO FEDERAL
Paraná
Campus Paranaguá

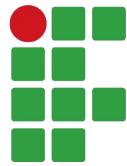
Hypertext Transfer Protocol

HTTP



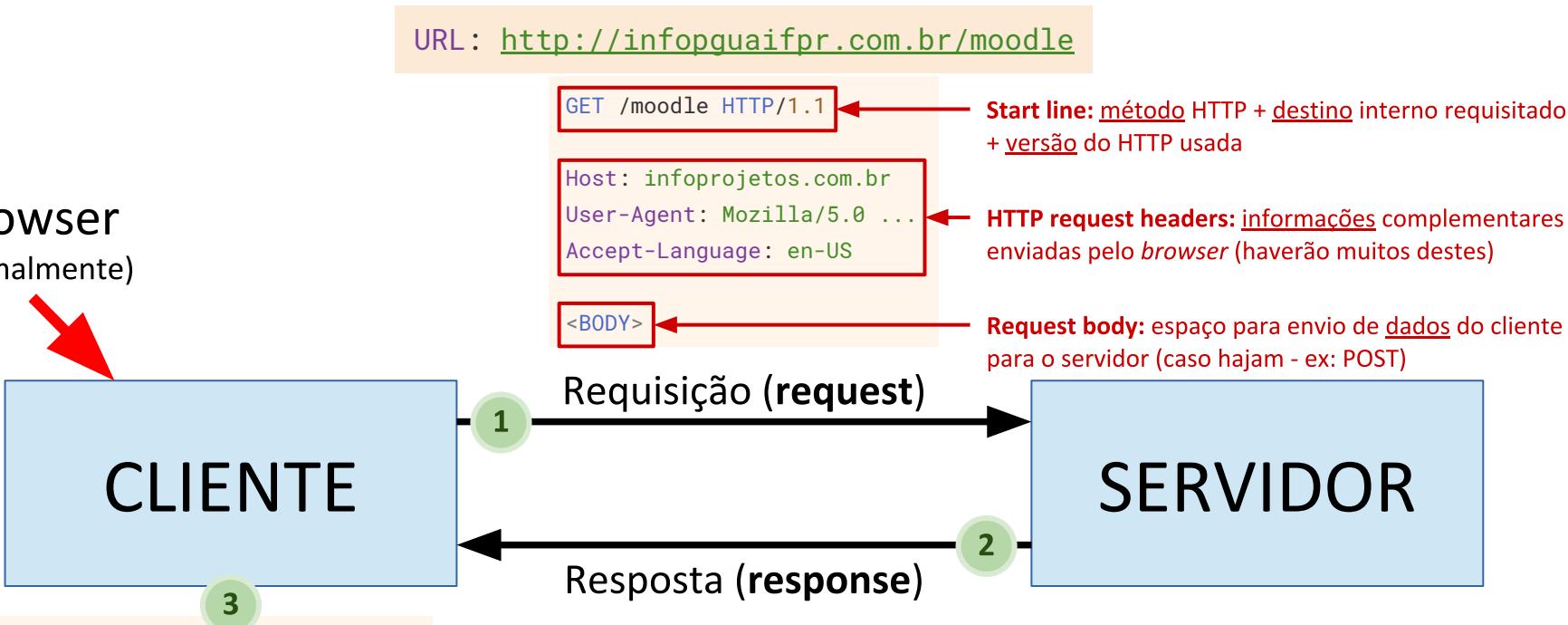
HTTP

- **Hypertext Transfer Protocol**
 - Protocolo de Transferência de Hipertexto
- É um protocolo de comunicação
- Executa na camada de aplicação (modelo OSI)
- Possibilitou o estabelecimento da Web
- Define como Cliente e Servidor trocam informações
 - Requisição e Resposta

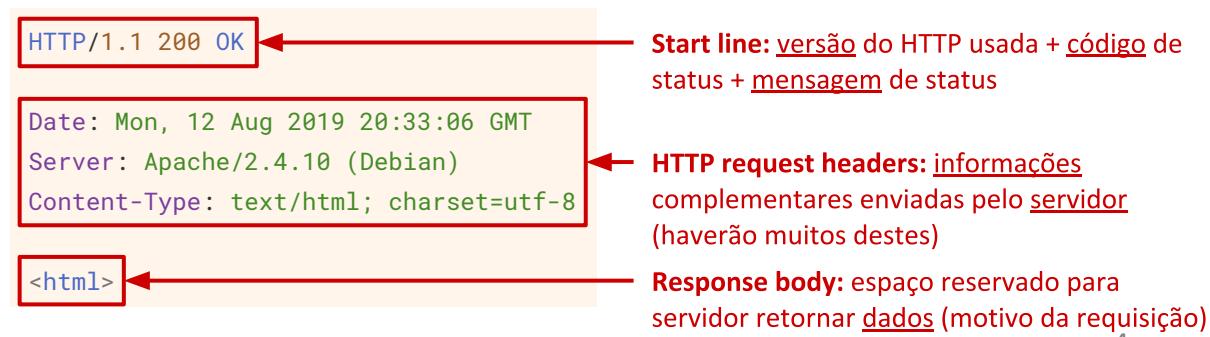


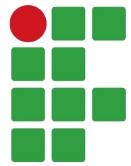
Comunicação HTTP

Browser
(normalmente)



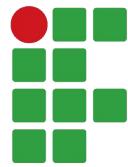
1. Browser recebe a resposta (index.html)
2. Começa a renderizar página
3. Escaneia recursos (JS, CSS, imagens, ...)
- Repete o processo para cada recurso





Método GET

- Parâmetros são enviados através da URL
 - Visíveis na barra de endereços (se existirem)
 - Chamamos eles de Query String
- URL + parâmetros têm o seguinte formato:
 - [URL]?parametro1=valor1¶metro2=valor2
 - Servidor sabe “desmontar” URL
- É muito utilizado em:
 - Digitação direta da URL
 - Hiperlinks
 - Consultas que não alteram dados no servidor



GET - Exemplos de URL

`http://localhost/MinhaApp?visivel=sim`

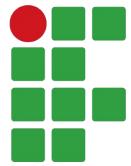
`http://www.exemplo.com?mostrar=tudo&filtrar=usuario`

`http://paranagua.ifpr.edu.br`

`http://127.0.0.1:8084?login=administrador&senha=123`

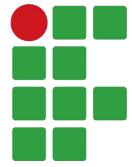
`http://www.meubanco.com/extrato?agencia=12345&conta=12345x&senha=98765432`

`http://www.algumsite.com.br/forum/viewtopic.php?t=71645&sid=1949d3e950afec7a19bb9ee7a37052f0`



Método POST

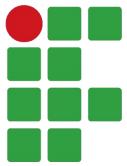
- Parâmetros não utilizam a URL
 - Não ficam visíveis na barra de endereços
 - Não quer dizer que não existam
- Para envio de texto puro ou outros (binários)
- É muito utilizado em:
 - Envio de formulários HTML
 - Upload de arquivos
 - Autenticação



Código de Status

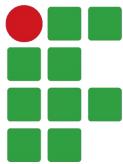
- Informa ao browser como (e se) a requisição atendida pelo servidor
 - 200 OK
 - 404 Not Found
 - ...
- Melhor explicado com cachorrinhos
 - <https://httpstatusdogs.com>





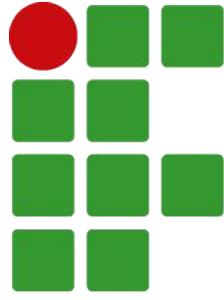
Nosso Servidor

- Implementamos nosso servidor com o **módulo http do Node.js puro**
- Vantagens
 - Facilidade
 - Direto ao ponto
 - Podemos criar rotas com um novo if
 - Total controle da requisição (*request*)
 - Total controle da geração da resposta (*response*)
 - ...



Problemas

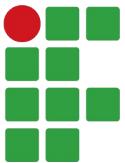
- Crescimento da aplicação
- Rotas (URLs) mais complexas
- Requisições com mais informações
 - Parâmetros, métodos HTTP, ...
- Respostas customizadas
 - HTML estático ou gerado com String
 - JSON, HTML, XML, ...
- Difícil separação de responsabilidades
 - Quem faz o quê?
- if else if, ao infinito e além



INSTITUTO FEDERAL
Paraná
Campus Paranaguá

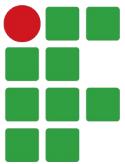
Fast, unopinionated, minimalist web framework for Node.js

Framework Express



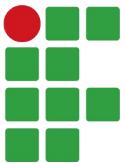
Express

- Framework web para Node.js
- Minimalista
- Node.js puro
- Maior abstração
- Muitas ferramentas
 - Rotas complexas
 - Manuseio de requisições e respostas mais fácil
 - Middlewares
 - Renderização de HTML no servidor
 - ...



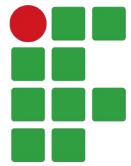
Express

- Proporciona produtividade
 - Não reinventar a roda
- Facilita a aplicação do Modelo MVC
 - Model-View-Controller
 - Separação de responsabilidades
- Desempenho na execução
 - É rápido por ser simples



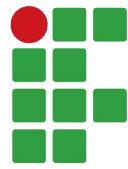
Express

- Site
 - <https://expressjs.com>
- Instalação
 - `npm i express@4 --save`
 - O @4 força a utilização da versão 4
 - Versão 5 está em beta e quando sair pode gerar incompatibilidade de trechos de código
 - Aplicação já com Node.js, NPM, nodemon, ...
- Versão atual
 - 4.17.1



Usando o Express

- Para começar a usar o Express, precisamos:
 - Criar **app.js** (escolher nome)
 - Importar (**require**)
 - Instanciar
 - Ouvir IP:porta
 - Registrar rotas válidas
 - URLs e métodos HTTP
 - Para cada rota:
 - Funções de callback
 - Receber e tratar requisições
 - Gerar e devolver respostas



app.js - Servidor Simples

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

const app = express();

Método HTTP GET
app.get('/', (req, res) => {
    res.status(200).send(`Hello Express Framework! Você acessou: ${req.url}`);
});

Retornando 200 OK (sucesso)

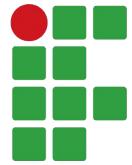
const port = 3000;
app.listen(port, () => {
    console.log(`Servidor rodando na porta ${port}.`);
    console.log(`Acesse http://localhost:${port} para testar.`);
});
```

Rota:
http://localhost:3000/

req (request) → dados do browser
res (response) → para gerar a resposta

Callback function

Resposta para o cliente (browser)



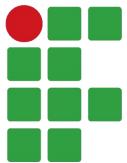
app.js - Servidor Simples

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

const app = express();

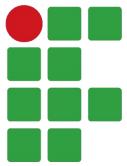
app.get('/', (req, res) => {
  res.status(200).send(`Hello Express Framework! Você acessou: ${req.url}`);
});

const port = 3000;
app.listen(port, () => {
  console.log(`Servidor rodando na porta ${port}.`);
  console.log(`Acesse http://localhost:${port} para testar.`);
});
```



Executando

- Para executar:
 - `nodemon app.js`
 - Verificar terminal
- Digitar no browser:
 - <http://localhost:3000>
 - Verificar resultado na página
 - Verificar terminal



Scripts NPM

- Criar script no package.json
 - Iniciar aplicação pelo NPM

Definindo o arquivo JS principal da aplicação

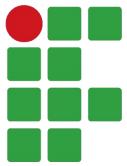
Lista de scripts da nossa aplicação

Nome do script: start

Comando a executar: nodemon .

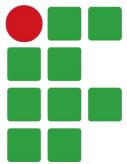
- Executar no terminal
 - `npm run start` ou
 - `npm start`

```
{  
  "name": "express-aula",  
  "version": "1.0.0",  
  "description": "",  
  "main": "app.js",  
  "scripts": {  
    "start": "nodemon ."  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.17.1"  
  },  
  "devDependencies": {  
    "...": "...",  
  }  
}
```



Ferramentas

- Instalar extensões e ferramentas de auxílio
- Abrir Extensions (Ctrl + Shift + X)
- Instalar:
 - **Prettier**
 - Formata diversos tipos de arquivo
 - **ESLint**
 - Mostra erros e más práticas de JavaScript
- Abrir tela de Preferências (Ctrl + ,)
 - Buscar “format on save”
 - Marcar Editor: Format On Save

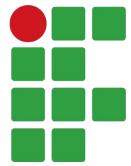


Ferramentas

- Instalar dependências de projeto
- Executar:

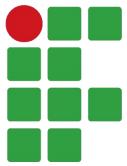
```
npm i eslint prettier eslint-config-prettier eslint-plugin-prettier
eslint-config-airbnb eslint-plugin-node eslint-plugin-import
eslint-plugin-import eslint-plugin-jsx-a11y eslint-plugin-react --save-dev
```

- Ver package.json
- Baixar arquivos de configuração do Moodle
 - ZIP com → .prettierrc .eslintrc.json
 - Configurações padrão do professor
 - Descompactar na raiz do projeto
- Reiniciar Visual Studio Code



Ferramentas - Aviso

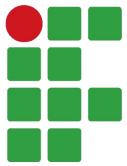
- As ferramentas instaladas vão opinar sobre nosso código
- **Opinar** significa:
 - Apontar erros, más práticas e possíveis melhorias
 - ESLint
 - Modificar de fato: formatar, adicionar trechos,
 - ...
 - Prettier
- Primeira impressão → **Poxa! que treco chato!**
 - Muitos benefícios a longo (e curto) prazo



Parâmetros

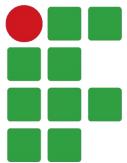
- Podemos ler os parâmetro da query string
 - ?param1=valor1¶m2=valor2
- Usar parâmetro da requisição
 - `req.query`
 - O Express já transforma em objeto JS para nós
- Podemos ler com:

```
console.log(req.query.param1); // Imprime "valor1" no console  
console.log(req.query.param2); // Imprime "valor2" no console
```



Parâmetros

- Os valores dos parâmetros sempre são Strings
 - Verificar necessidade de eventual conversão
- E se eu pedir por um parâmetro que não está definido na URL?
 - Você receberá um **undefined**

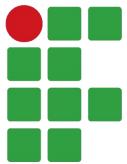


Exemplo

- Pede a idade e uma quantidade de anos
 - Informa a idade da pessoa após estes anos

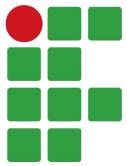
// URL --> <http://localhost:3000/calcularIdade?idade=31&anos=5>

```
app.get('/calcularIdade', (req, res) => {
  const idade = parseInt(req.query.idade, 10);
  const quantidadeAnos = parseInt(req.query.anos, 10);
  const idadeAposAnos = idade + quantidadeAnos;
  res
    .status(200)
    .send(`Daqui a ${quantidadeAnos} anos você terá ${idadeAposAnos} anos.`);
});
```



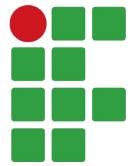
Prática

- Amplie o exemplo anterior
- A URL também pode conter um nome
- Verificar no servidor
 - Tem nome na URL?
 - Olá Nome! Daqui a X anos você terá Y anos.
 - Não tem?
 - Olá anônimo! Daqui a X anos você terá Y anos.
 - Caso não haja idade ou anos na URL:
 - Não foi possível calcular a sua idade.
 - Status 404



Tipo do Retorno

- Estamos retornando texto puro
 - Não é legal (o browser espera algo melhor) 😊
- O ideal é retornar HTML
 - Espere! Logo mais!
- Por ora, podemos facilmente retornar JSON
 - Resultado melhor formatado 👍
 - Faz mais sentido
 - Lembra das APIs?

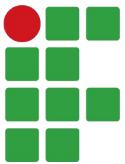


Retornando JSON

// URL --> <http://localhost:3000/calcularIdade?idade=31&anos=5>

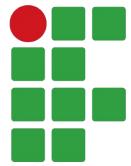
```
app.get('/calcularIdade', (req, res) => {
  const idade = parseInt(req.query.idade, 10);
  const anos = parseInt(req.query.anos, 10);
  const idadeAposAnos = idade + anos;
  const mensagem = `Daqui a ${anos} anos você terá ${idadeAposAnos} anos.`;
  res.status(200).json({
    status: 'success',
    message: mensagem
  });
});
```

Estrutura de JSON para retorno
de APIs bastante difundida
[JSend](#)



Prática

- Refatore a prática do cálculo de idade
- Retornar dados em JSON
 - Utilizar formato JSend
 - Não esquecer de customizar status
 - Não esquecer do status code do HTTP



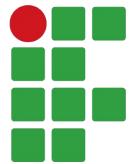
Requisição POST

- Também conseguimos receber requisições via método POST
 - Parâmetros não vêm pela URL
 - Eles vêm “escondidos” no corpo da requisição
- Para ler:
 - Adicionar no começo do código:

```
app.use(express.urlencoded({ extended: false }));
```

- Na requisição, usar atributo **body**

```
console.log(req.body.parametro);
```

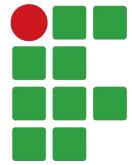


Exemplo - Login

```
const express = require('express');
const app = express();
app.use(express.urlencoded({ extended: false })); // IMPORTANTE
// ... Várias outras rotas ocultas ...

// URL --> http://localhost:3000/login
app.post('/login', (req, res) => {
    let statusCode, statusMessage, mensagem;
    if (req.body.email === 'aluno@ifpr.edu.br' && req.body.senha === '123456') {
        statusCode = 200;
        statusMessage = 'success';
        mensagem = 'Usuário logado com sucesso';
    } else {
        statusCode = 401;
        statusMessage = 'fail';
        mensagem = 'Login ou senha inválidos';
    }
    res.status(statusCode).json({
        status: statusMessage,
        message: mensagem
    });
});

// ... Mais código oculto ...
```

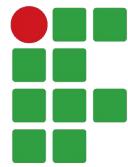


Requisição POST

- Tá! E como faço uma requisição agora?
- URL: <http://localhost:3000/login>
- E os parâmetros?
- Como mudo o método HTTP?
- O “certo” seria usar um formulário HTML
 - Mas ainda não estamos gerando HTML
- Então?
 - [Postman](#)



POSTMAN



Requisição com Postman

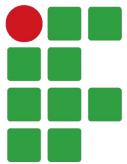
The screenshot shows the Postman interface with the following annotations:

- Método HTTP**: Points to the "POST" button in the top left.
- URL**: Points to the "http://localhost:3000/login" input field.
- Envia dados no corpo da requisição**: Points to the "Body" tab.
- Fazer requisição**: Points to the "Send" button.
- Simula submissão de formulário HTML**: Points to the "x-www-form-urlencoded" radio button under the Body tab.
- Parâmetros**: Points to the "email" and "senha" parameters in the "Body" table.
- Resposta**: Points to the JSON response in the "Pretty" tab.

Postman interface details:

- Request Method: POST
- Request URL: http://localhost:3000/login
- Request Type: Untitled Request
- Headers: (10)
- Body: (x-www-form-urlencoded)
- Params: (none)
- Authorization: (none)
- Headers: (10)
- Tests: (none)
- Cookies: (none)
- Code: (none)
- Comments: (0)
- Body (Pretty):

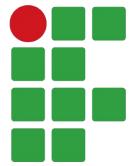
```
1 {  
2   "status": "success",  
3   "message": "Usuário logado com sucesso"  
4 }
```
- Status: 200 OK
- Time: 12ms
- Size: 272 B
- Save Response



Atividade

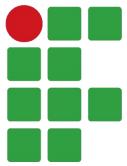
- Você já sabe carregar um arquivo de texto em uma variável String
 - Certo?
- Baixe o arquivo usuarios.json do Moodle
- Carregue o arquivo JSON como String
- Código que converte String para objeto JS:

```
const usuarios = JSON.parse(dadosDoSeuArquivoJsonEmString);
```



Atividade - Rota 1

- Faça a sua rota **POST /login** validar o email e senha com base nos usuários carregados do arquivo JSON
 - Responder o mesmo JSON com JSend de antes
 - Mudar mensagem de sucesso para:
 - “Seja bem vindo, NOME!”



Atividade - Rota 2

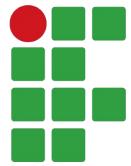
- Crie uma nova rota **GET /usuarios**
 - Ela deve retornar todos os usuários do JSON

Pode ser assim. Mais fácil.
(informações sensíveis)

```
{  
  "status": "success",  
  "results": 2,  
  "data": {  
    "usuarios": [  
      {  
        "id": 1,  
        "nome": "Fritz",  
        "email": "fritz@caozinho.com",  
        "senha": "123caozinho"  
      },  
      {  
        "id": 2,  
        "nome": "Franz",  
        "email": "franz@raivoso.com",  
        "senha": "puroódio"  
      }  
    ]  
  }  
}
```

```
{  
  "status": "success",  
  "results": 2,  
  "data": {  
    "usuarios": [  
      {  
        "id": 1,  
        "nome": "Fritz"  
      },  
      {  
        "id": 2,  
        "nome": "Franz"  
      }  
    ]  
  }  
}
```

DESAFIO



Atividade - Rota 3

- Crie uma nova rota **GET /usuario**
 - Elas deve aceitar um parâmetro id (obrigatório)
 - Exemplo: **/usuario?id=5**
- Retornos válidos:

```
{  
    "status": "fail",  
    "message": "Usuário com ID 54654 não encontrado",  
    "data": null  
}  
  
{  
    "status": "success",  
    "data": {  
        "usuario": {  
            "id": 1,  
            "nome": "Fritz"  
        }  
    }  
}  
  
{  
    "status": "fail",  
    "message": "Informe um ID",  
    "data": null  
}
```