

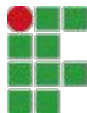
Plano de Aula IFPR

<https://github.com/lgapontes/ifpr>



+

express

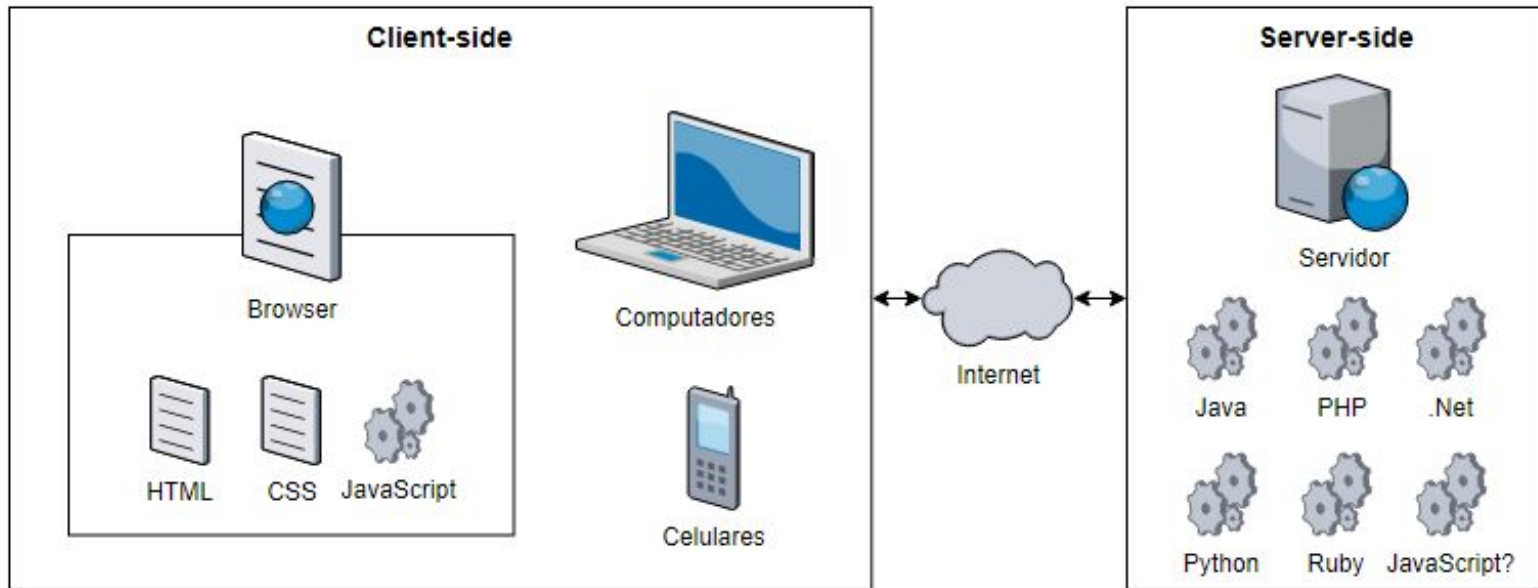


**INSTITUTO
FEDERAL**
Paraná

Configuração do NodeJS com Express

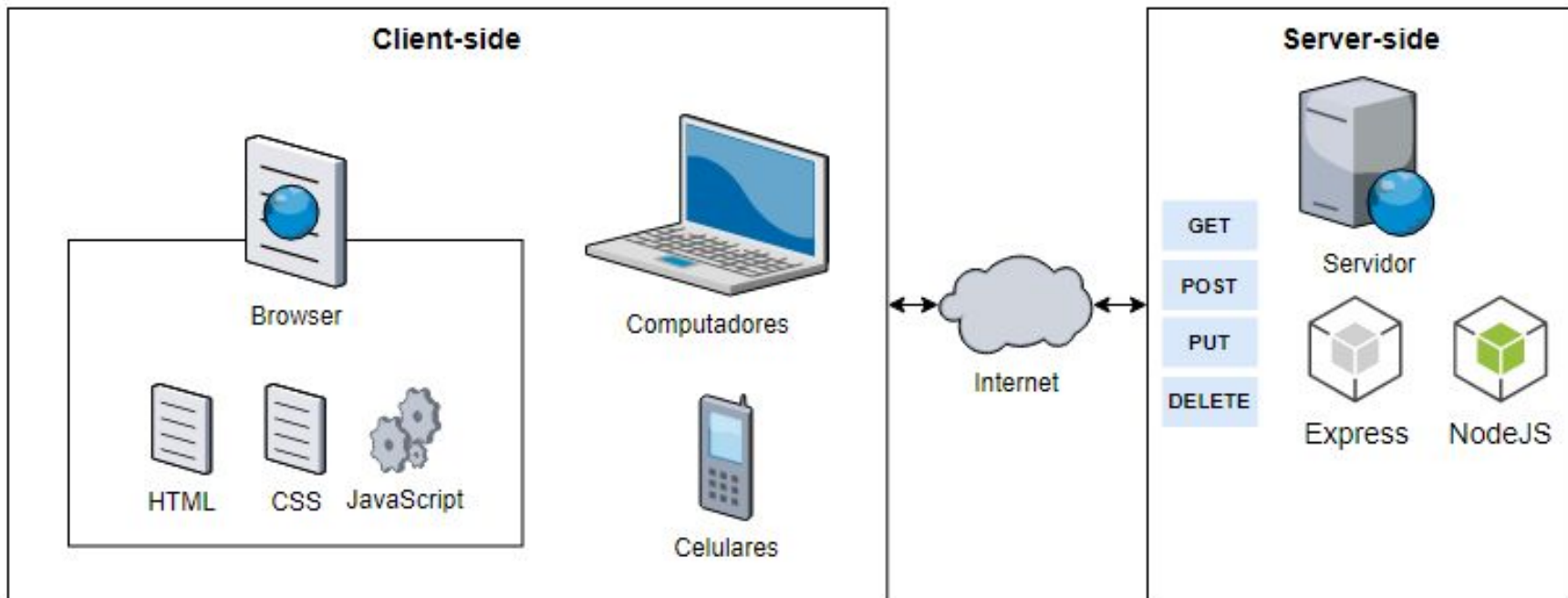
NodeJS e NPM

Node.js é um runtime JavaScript open-source e cross-platform que executa código (JavaScript) no backend. **NPM** (Node Package Manager) é um gerenciador de pacotes para Node. Ele já vem instalado.



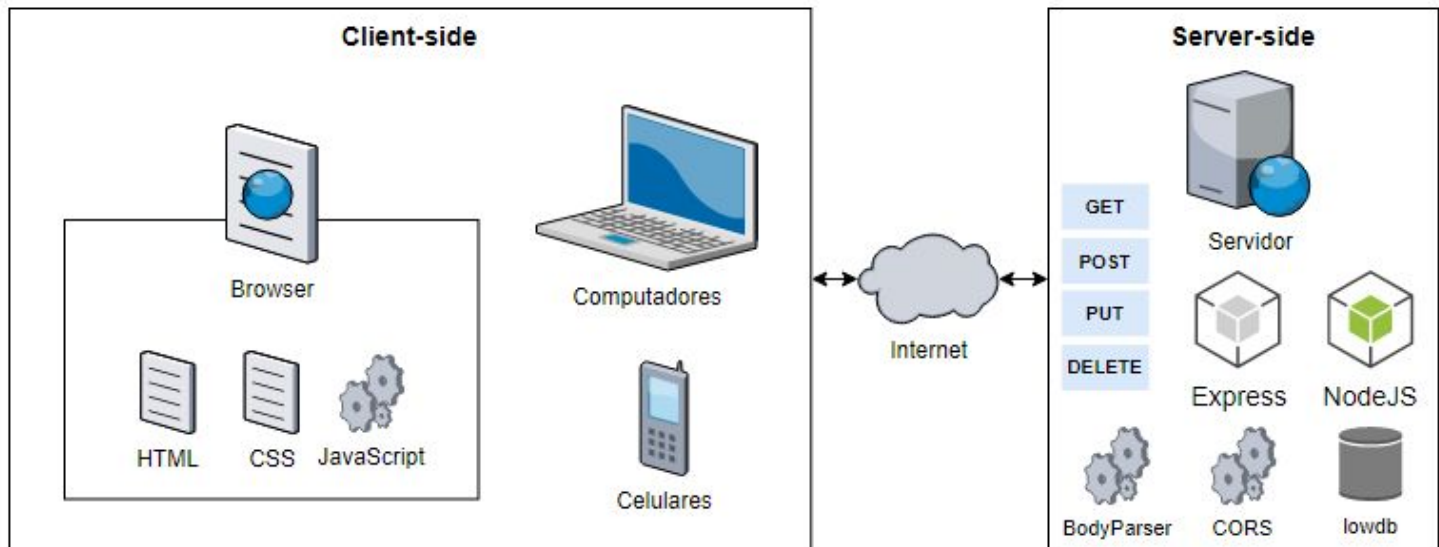
Express

Express (ou **Express.js**) é um dos mais populares frameworks de aplicações web para Node.js, muito utilizado para construção de aplicações e API's.



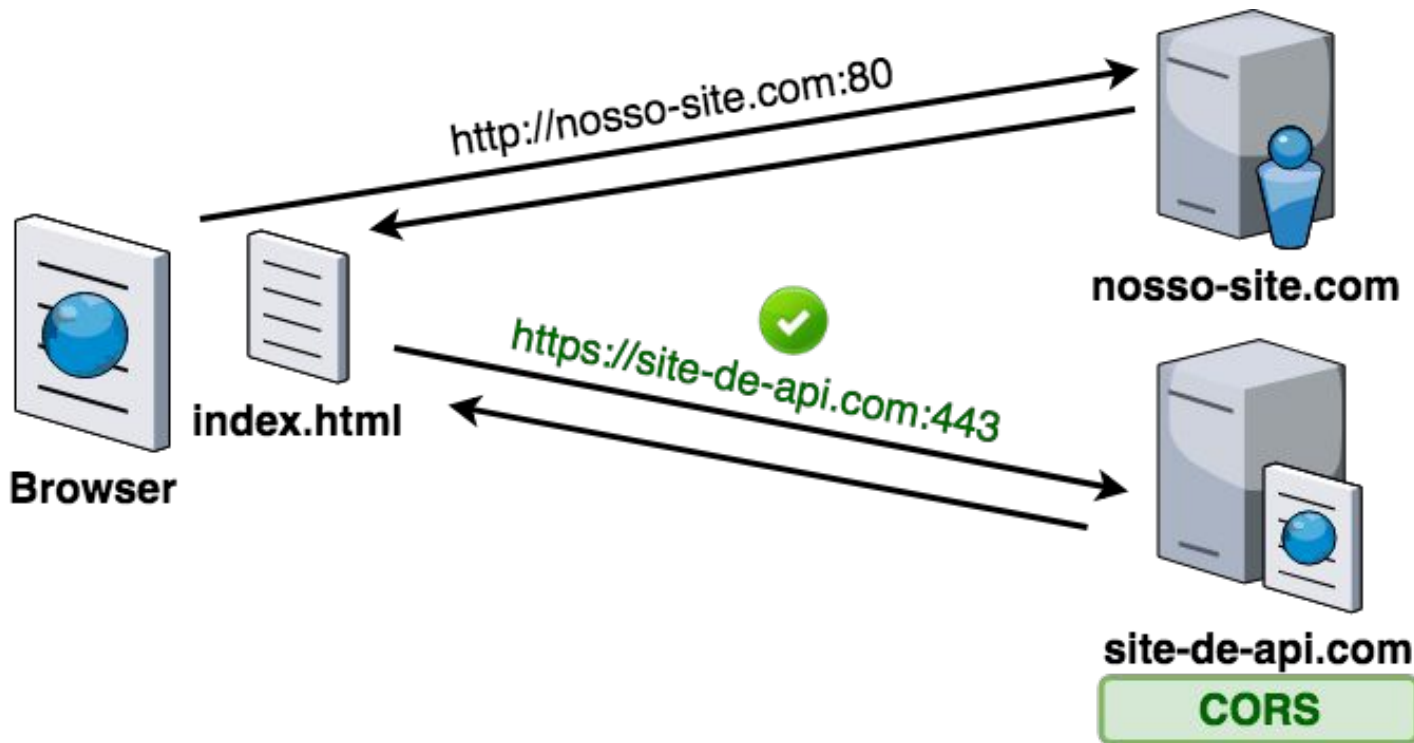
BodyParser, lowdb, CORS

O **body-parser** é um módulo capaz de converter o body da requisição para vários formatos, inclusive json. **lowdb** é um pequeno banco JSON para Node (vamos utilizá-lo para representar a etapa de persistência). **CORS** é um módulo para habilitar o Cross-origin resource sharing.



Cross-Origin Resource Sharing (CORS)

O CORS é um recurso configurado no **servidor** para avisar aos browsers que determinado site **aceita** requisições de outras origens.



Instalando, configurando e executando

PASSO 1: Baixar o node e instalar no SO

<https://nodejs.org/en/download/>

PASSO 2: Iniciar projeto e instalar os pacotes pelo NPM

```
npm init
```

```
npm install express body-parser cors lowdb uuid
```

PASSO 3: Criar um arquivo chamado exemplo.js

```
var express = require('express')
var cors = require('cors')
var app = express()
app.use(cors())

app.get('/afazeres', function (req, res, next) {
  res.json([{afazer: 'Lavar o carro!'}])
})

app.listen(80, function () {
  console.log('http://127.0.0.1:80/afazeres')
})
```

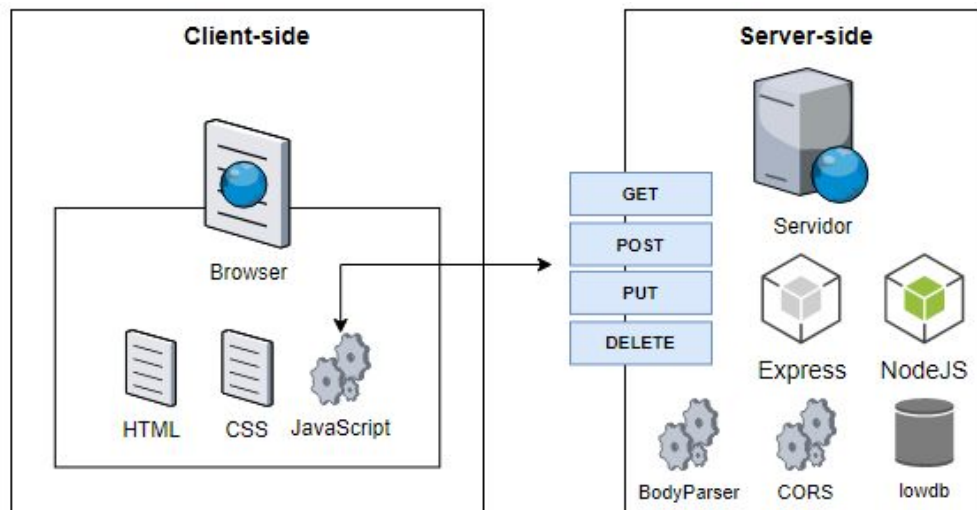
PASSO 4: Executar o script

```
node exemplo.js
```

Utilizando Web API semântica REST

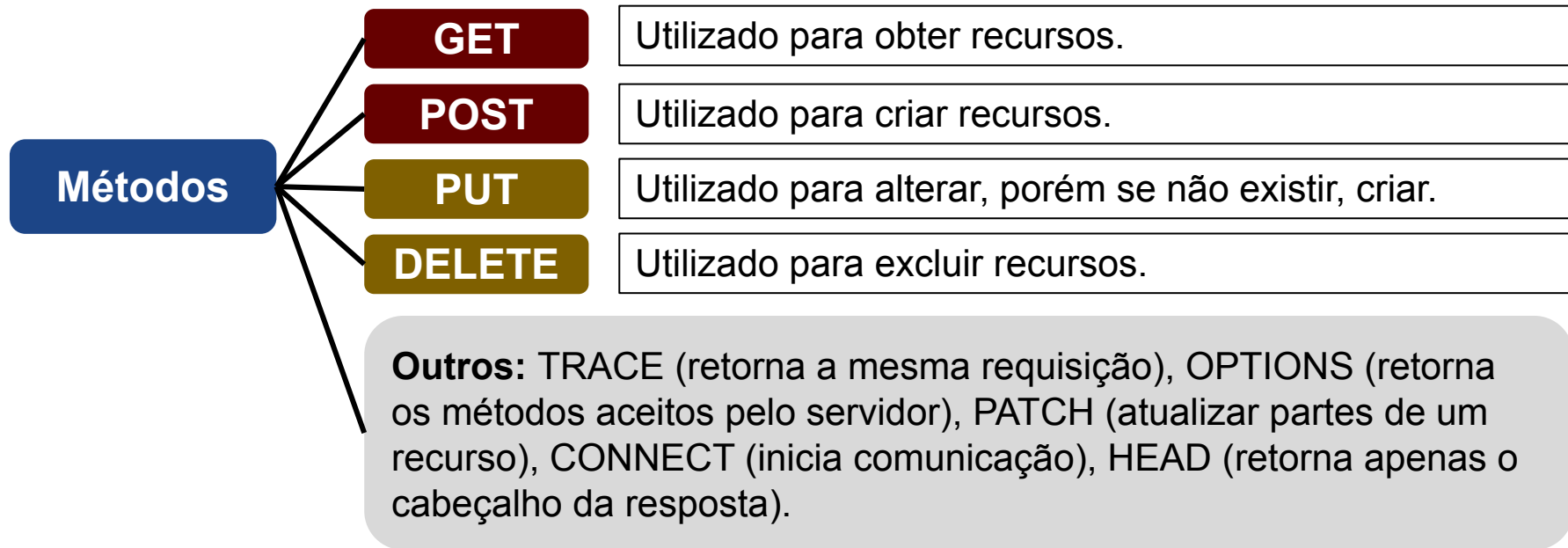
API REST

O acrônimo REST (REpresentational State Transfer) é uma arquitetura que define uma série de regras para construir uma API (Application Programming Interface) de serviços web baseados no protocolo HTTP. A principal ideia é utilizar-se dos métodos disponíveis no protocolo (por exemplo, o método GET) para viabilizar o tráfego de dados (em JSON).





Métodos da API

O protocolo HTTP oferece métodos que são facilmente ajustáveis às operações comuns de interoperabilidade entre sistemas.



Utilizando métodos com semântica

Semântica

- API em torno do recurso (<https://site-de-api.com/clientes>)
 - API baseada em substantivos (não em verbos)
 - Um request REST é atômico e stateless
 - Usar URI para manipular os recursos (ao invés de querystring)
-  /api/empregado/1029  /api/empregado?matricula=1029

Idempotência

Métodos que podem ser chamados várias vezes sem causar problemas no servidor. Quais? GET, OPTIONS, HEAD, PUT, TRACE, CONNECT e DELETE

Verbos

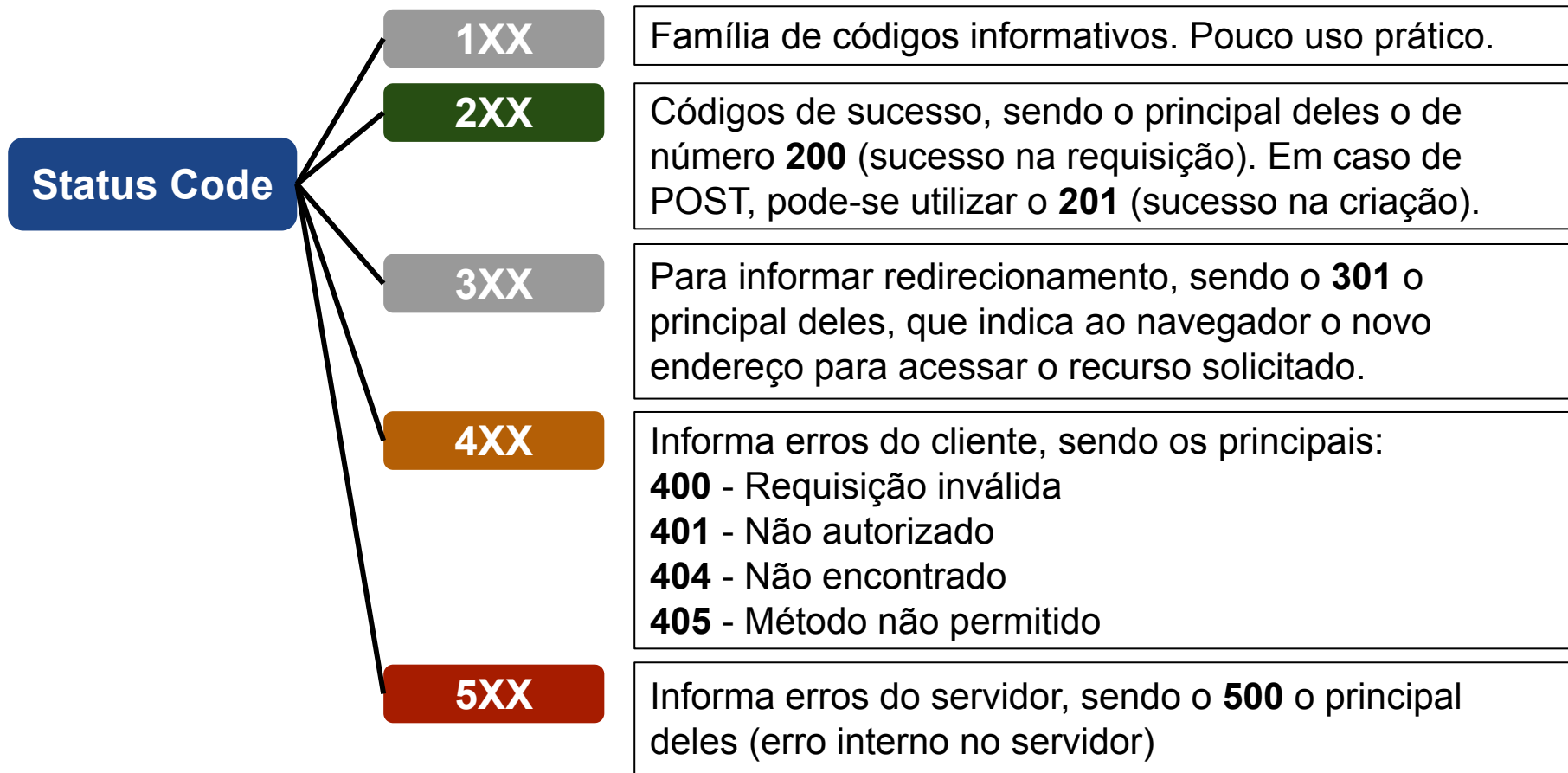
Deve-se utilizar GET para obter dados, POST para criar novos dados, etc.

Status code

Aplicar corretamente o Status Code do HTTP para trabalhar com as respostas dos serviços (exemplo no próximo Slide).

Códigos de status do HTTP

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

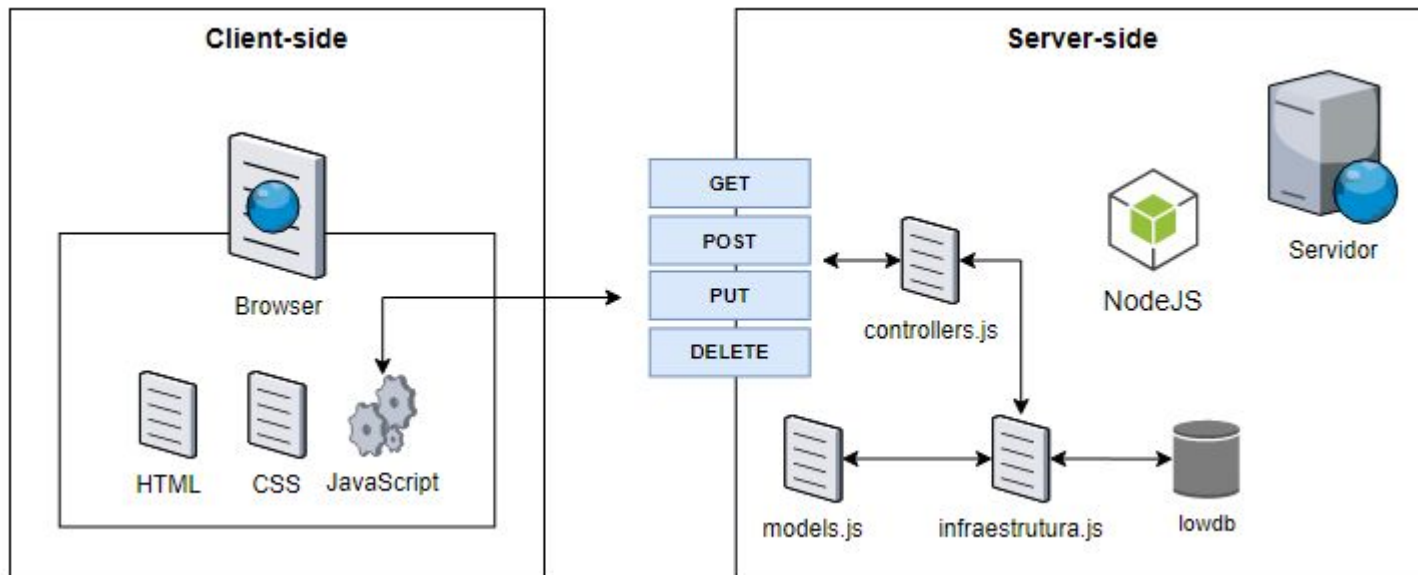


Criando um software completo

(Lista de Afazeres)

Camadas Controllers, Models, Infraestrutura

É uma boa prática organizar o código backend em Controllers (responsáveis pela recepção e retorno das solicitações), Models (entidades de negócio) e Infraestrutura (métodos de acesso aos recursos da aplicação).



Métodos GET, POST, PUT, DELETE

Devemos separar três rotas da URL do sistema para o recurso, cada uma com os métodos HTTP adequados.

<http://127.0.0.1/afazerres>

GET - Obter todos os registros de afazerres. Se não encontrar, retornar um array vazio.

<http://127.0.0.1/afazer/id>

GET - Obter um registro pelo ID informado na URL. Se não encontrar, retornar um 404.

PUT - Salvar um JSON enviado pelo body da requisição no registro identificado pelo ID da URL. Se não encontrar o registro, retornar uma 404. Se o registro for inválido, retornar um 400.

DELETE - Excluir um registro identificado pelo ID da URL. Se não encontrar o registro, retornar um 404.

<http://127.0.0.1/afazer>

POST - Salvar um novo registro. Se o registro estiver inválido, retornar um 400.

```
app.route('/afazerres')
  .get(function (req, res) {
    // ...
  })

app.route('/afazer/:id')
  .get(function (req, res) {
    // req.params.id ...
  })
  .put(function (req, res) {
    // req.params.id ...
  })
  .delete(function (req, res) {
    // req.params.id ...
  })

app.route('/afazer')
  .post(function (req, res) {
    // req.body ...
  })
```

Arquivos do backend



`controllers.js`

Arquivo para organizar os controllers do projeto, identificando a solicitação e realizando a ação necessária. Retornará para o cliente dados e statusCode de 200 (em caso de sucesso), 400 (em caso de solicitação inválida) ou 404 (caso o registro não seja encontrado).



`models.js`

Arquivo para controlar o modelo da solução. Além dos dados de negócio, é interessante implementar regras de validação e métodos para auxiliar no uso do modelo (tornando-o um modelo “rico”).



`infraestrutura.js`

Arquivo para controlar o acesso aos recursos de infraestrutura da aplicação. Neste sistema ele será o responsável por acessar o banco de dados lowdb.

Implementando cliente

Atualizar Afazeres

```
// Atualizar
fetch('http://127.0.0.1/afazer/' + afazer.uuid, {
  headers: { "Content-Type": "application/json; charset=utf-8" },
  method: 'PUT',
  body: JSON.stringify(afazer)
}).then(response => {
  if (response.status == 400) {
    exibirMensagem('Afazer deve ser um texto com no máximo 20 caracteres!');
  } else if (response.status == 404) {
    exibirMensagem('Registro não encontrado!');
  } else {
    esconderMensagem();
    exibirAfazeres();
    limparForm();
  }
});
```

Apagar Afazeres

```
// Apagar
fetch('http://127.0.0.1/afazer/' + uuid, {
  method: 'DELETE'
}).then(response => {
  if (response.status == 404) {
    exibirMensagem('Registro não encontrado!');
  } else {
    exibirMensagem('Registro excluído com sucesso!');
    exibirAfazeres();
  }
});
```

Inserir Afazeres

```
// Inserir
fetch('http://127.0.0.1/afazer', {
  headers: { "Content-Type": "application/json; charset=utf-8" },
  method: 'POST',
  body: JSON.stringify(afazer)
}).then(response => {
  if (response.status == 400) {
    exibirMensagem('Afazer deve ser um texto com no máximo 20 caracteres!');
  } else {
    esconderMensagem();
    exibirAfazeres();
    limparForm();
  }
});
```

Consultar Afazeres

```
// Consultar
fetch('http://127.0.0.1/afazeres')
.then(response => response.json())
.then(data => {
  let ul = document.querySelector('ul.consulta');
  ul.innerHTML = '';

  if (data.length == 0) {
    registroNaoEncontrado(ul);
  } else {
    data.forEach(afazer => {
      exibirAfazer(afazer, ul);
    });
  }
});
```

Arquivos do frontend



Neste exemplo, este arquivo contém as regras de front-end (JavaScript), a estilização (CSS) e a estrutura da página (HTML). Normalmente separamos tais elementos em arquivos segregados, mas para este caso (um exemplo pequeno), isso não se fez necessário.

Obrigado!