

GEX613 – Programação II

Modelo Cliente-servidor



1100/1101 – CIÊNCIA DA COMPUTAÇÃO

Giancarlo Salton & Edimar Junior

Modelo Cliente-Servidor

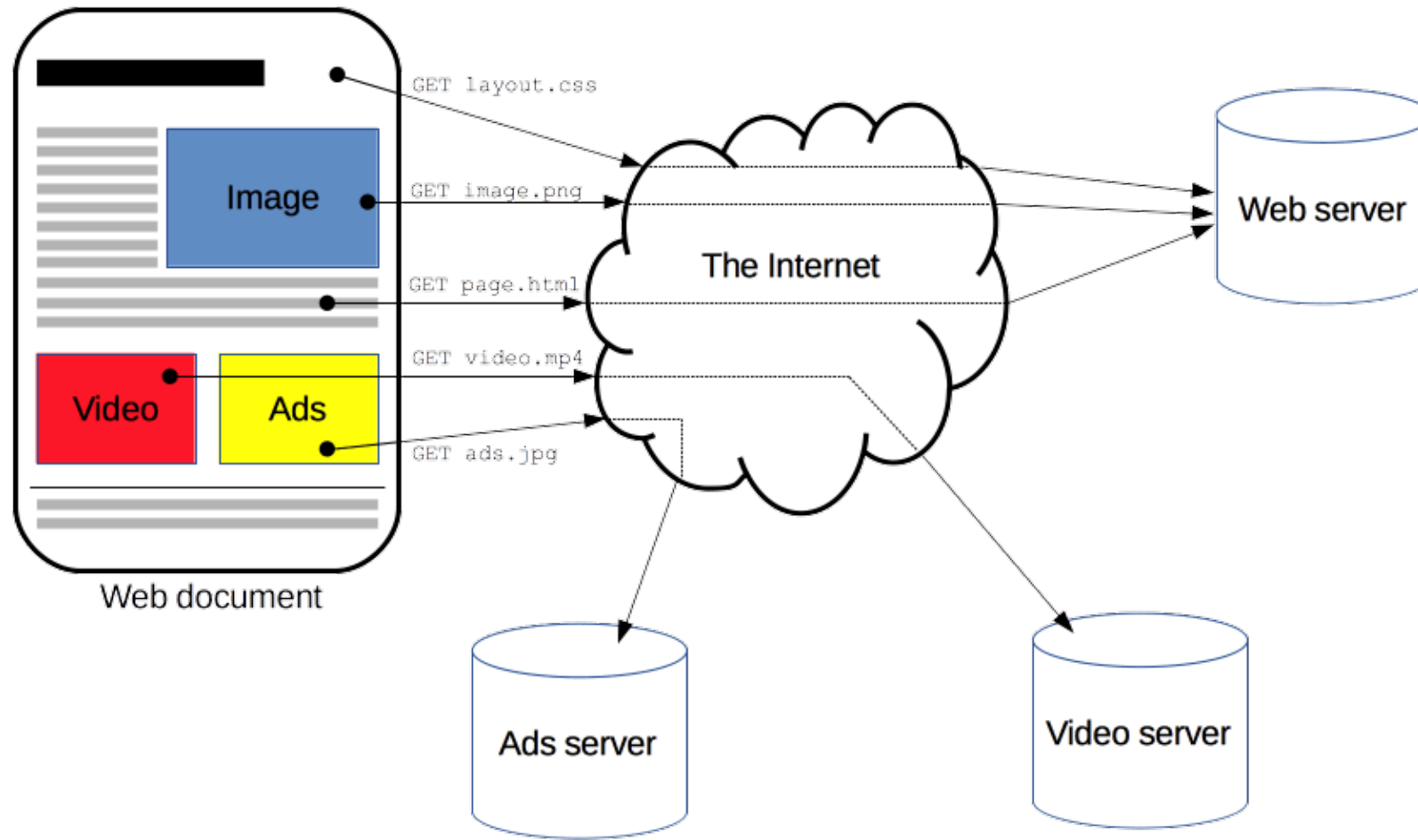
Protocolo HTTP

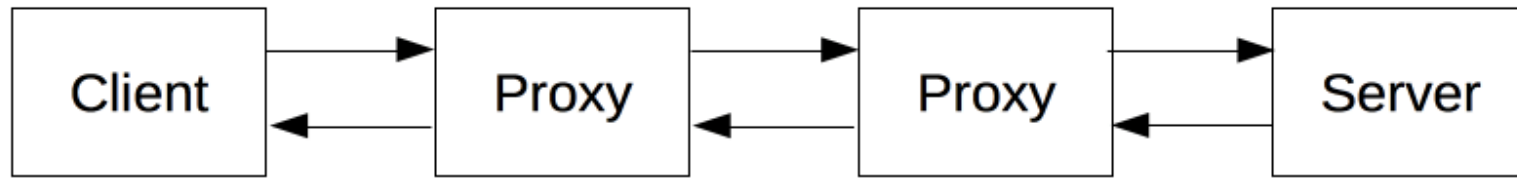
Endpoints & APIs REST

RESTfull

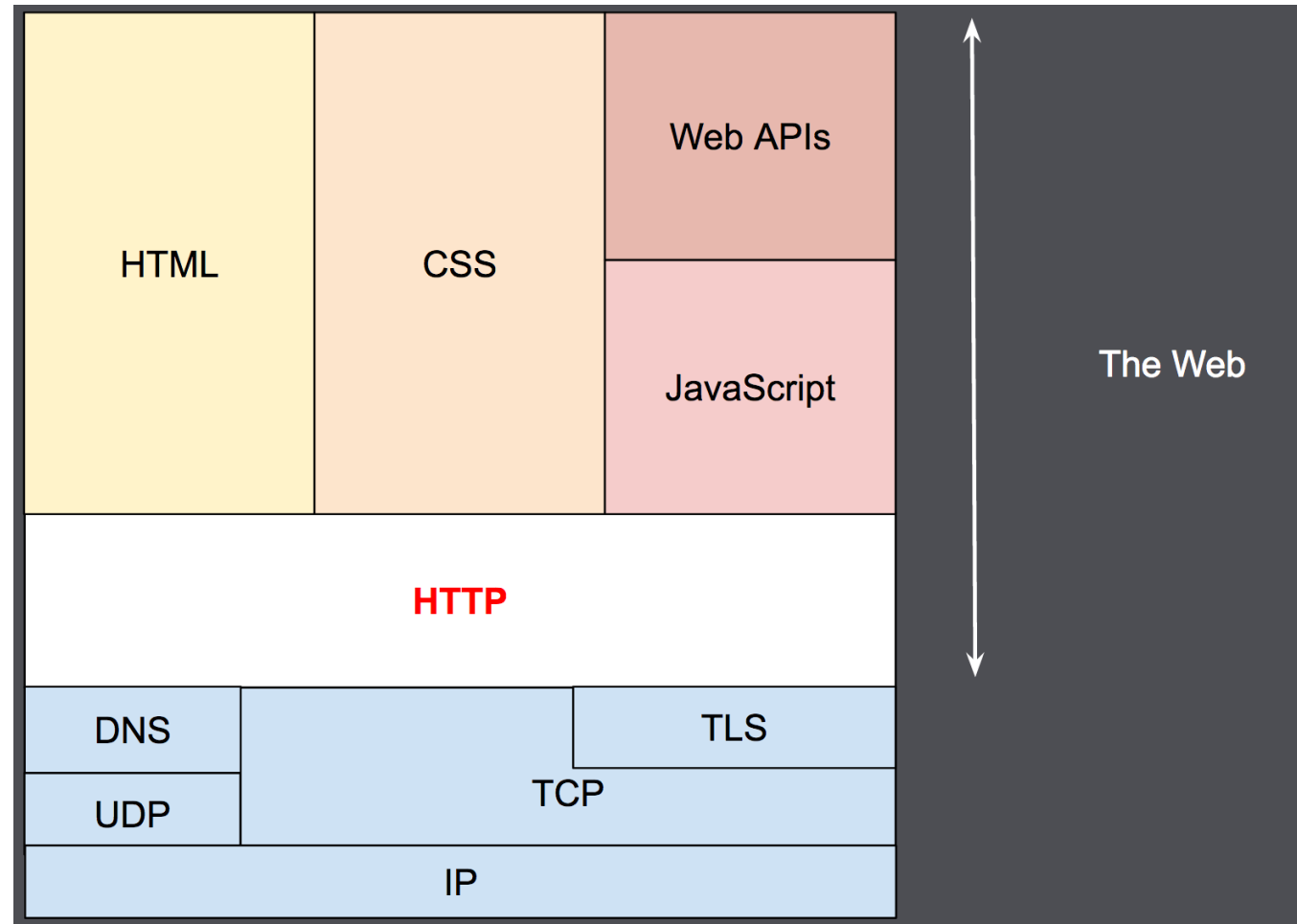
Express

Modelo Cliente-Servidor





Protocolo HTTP

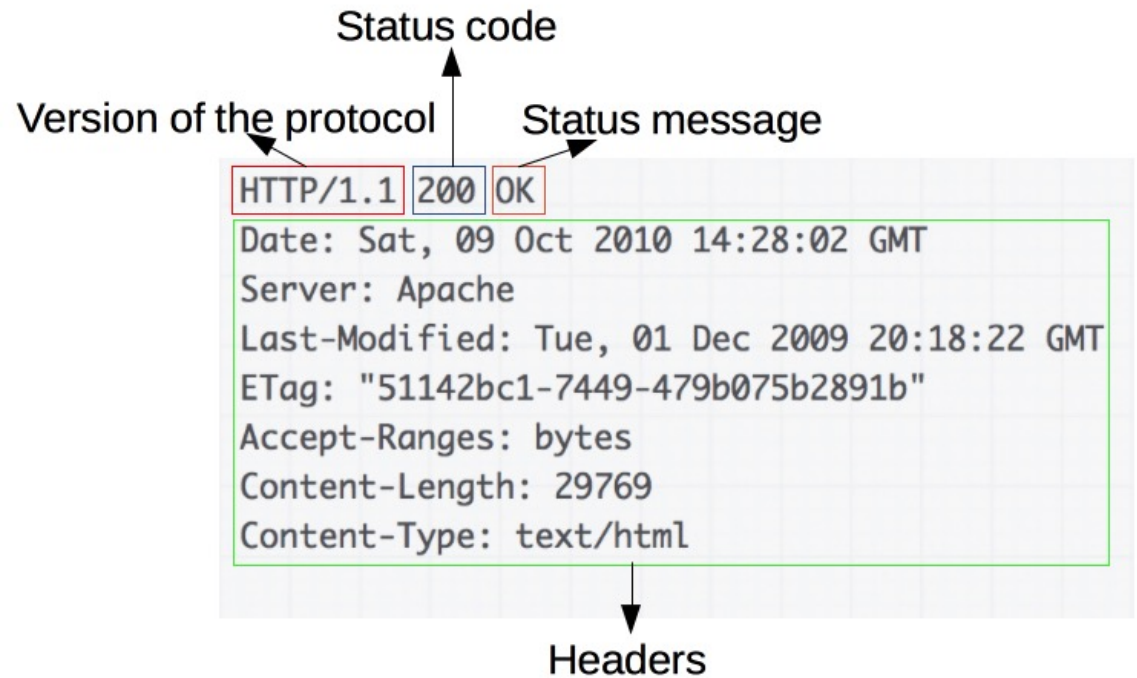
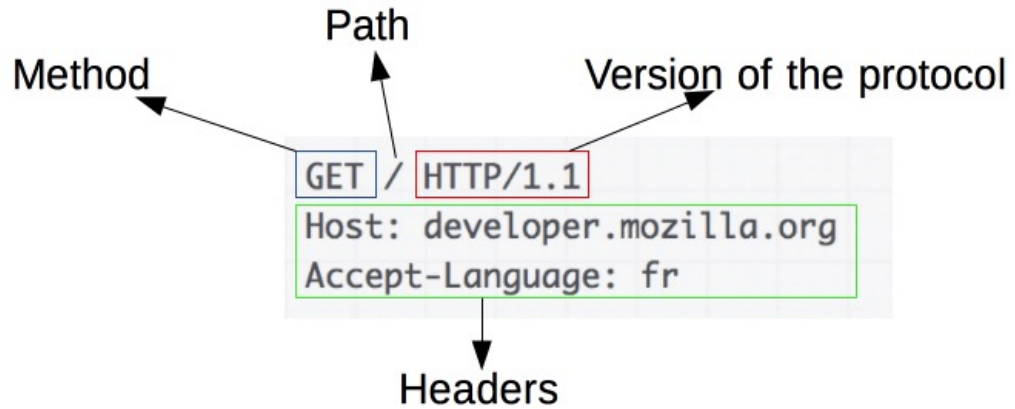


```
GET / HTTP/1.1  
Host: developer.mozilla.org  
Accept-Language: fr
```

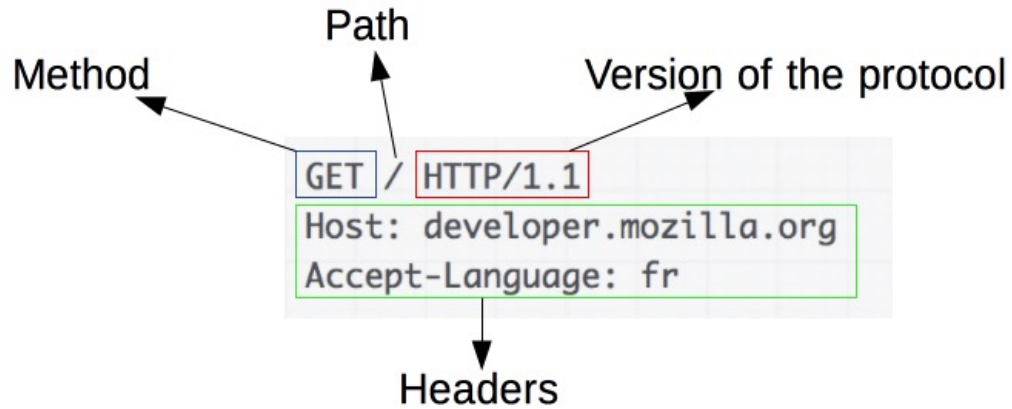
```
HTTP/1.1 200 OK  
Date: Sat, 09 Oct 2010 14:28:02 GMT  
Server: Apache  
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT  
ETag: "51142bc1-7449-479b075b2891b"  
Accept-Ranges: bytes  
Content-Length: 29769  
Content-Type: text/html
```

```
<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)
```

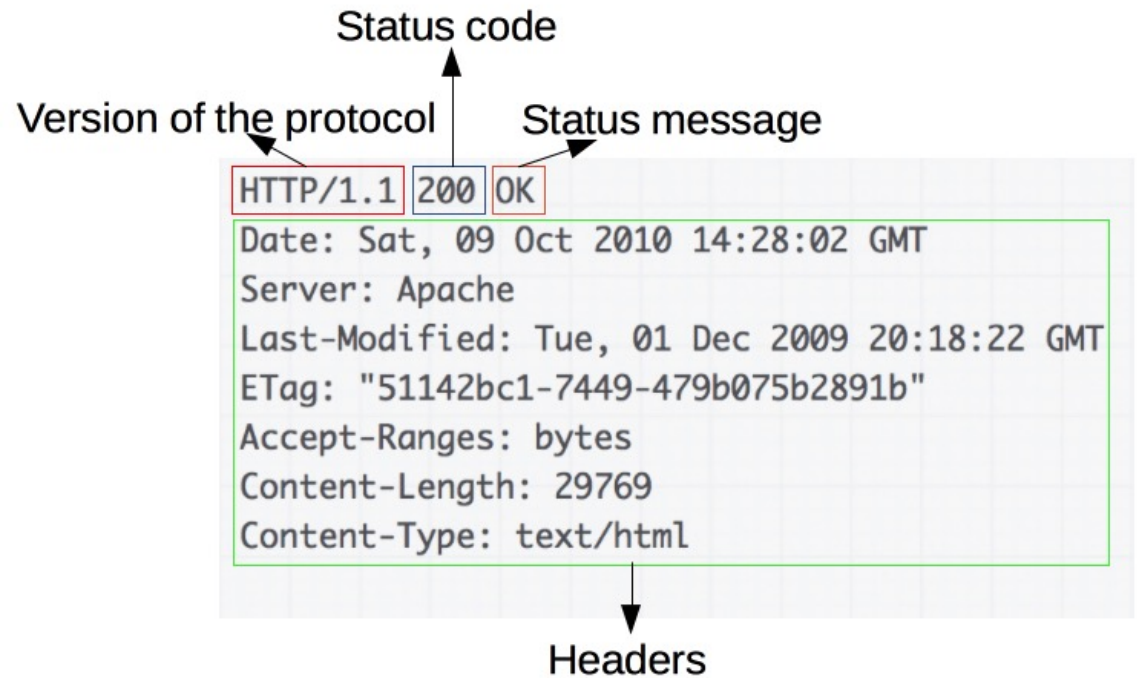

Request/Response



Request/Response



<https://httpstatusdogs.com/>



- GET

O método GET solicita a representação de um recurso específico. Requisições utilizando o método GET devem retornar apenas dados.

- POST

O método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.

- PUT

O método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.

- DELETE

O método DELETE remove um recurso específico.

- HEAD

O método HEAD solicita uma resposta de forma idêntica ao método GET, porém sem conter o corpo da resposta.

- CONNECT

O método CONNECT estabelece um túnel para o servidor identificado pelo recurso de destino.

- OPTIONS

O método OPTIONS é usado para descrever as opções de comunicação com o recurso de destino.

- TRACE

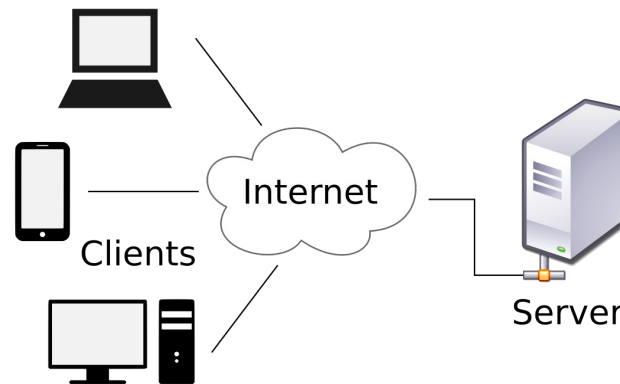
O método TRACE executa um teste de chamada *loop-back* junto com o caminho para o recurso de destino.

- PATCH

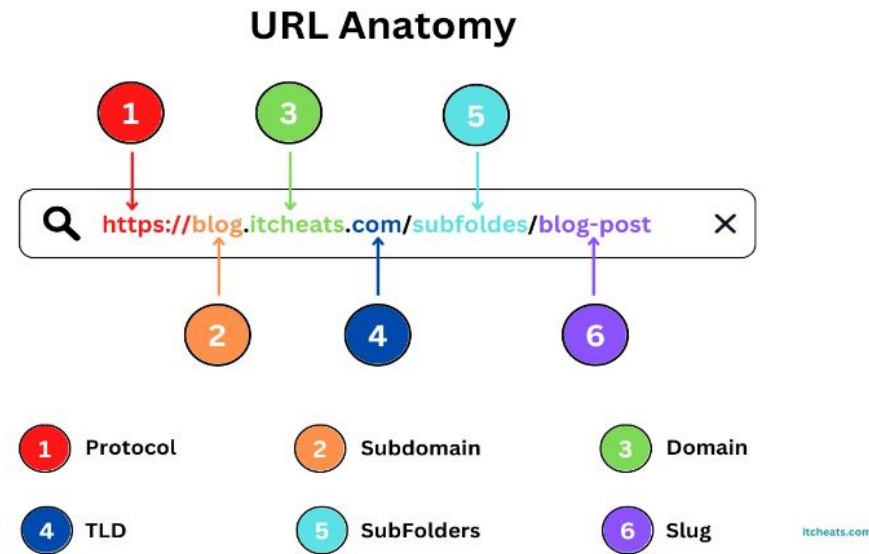
O método PATCH é utilizado para aplicar modificações parciais em um recurso.

Endpoints & APIs REST

- Um *endpoint* é um “local” onde uma requisição HTTP emitida pelo cliente chega no servidor
- O servidor processa a requisição e envia
- Cada *endpoint* é representado por uma URL
- A URL é convertida em endereço IP (*Internet Protocol*), utilizando um Servidor DNS (*Domain Name System*).



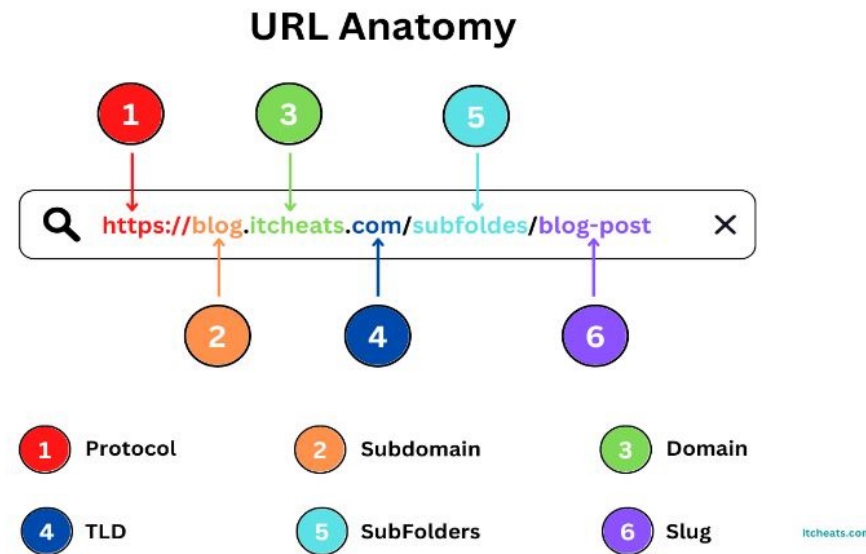
URL – Uniform Resource Locator



<http://www.uffs.edu.br/cc/programacao2/aula01.pdf>

Protocolo Domínio Caminho Recurso

URL – Uniform Resource Locator



<http://www.uffs.edu.br/cc/programacao2/aula01.pdf>

Protocolo Domínio Caminho Recurso

Um recurso pode ser uma chamada em uma API, como um POST ou GET.

- **RE**presentational **S**tate **T**ransfer
- Proposto como arquitetura em 2000 por Roy Fielding
- Define restrições a serem aplicadas sobre serviços-Web/APIs denominados REST que implementam interoperabilidade entre sistemas na Internet
- Utiliza os mesmos métodos do HTTP

Uma API que implementa todos os métodos do HTTP utilizando arquitetura REST é chamado de *RESTful*

- Modelo Cliente-Servidor
- Statelessness (sem estado)
- Cacheability
- Sistema multicamada
- Interface uniforme
- Code-on-demand (opcional)

JavaScript Object Notation (JSON)

- Formato de troca de dados, utilizado para comunicação entre APIs

Substitui o XML

- É uma sintaxe para serialização de objetos, matrizes, números, strings, booleanos, e null.

Baseia-se em sintaxe Javascript, mas é distinta desta: alguns Javascript não são JSON, e alguns JSON não são Javascript.

- Baseado em arquivos chave:valor

Exemplo:

```
let person = { name: "John", age: 31, city: "New York" }
```

Acesso:

```
person.name  
person[ "name" ]
```

Express

Criando um projeto Node.JS

`npm init`

→ `servidor` `npm init`

This utility will walk you through creating a package.json file.

It only covers the most common items, and tries to guess sensible defaults.

See ``npm help init`` for definitive documentation on these fields and exactly what they do.

Use ``npm install <pkg>`` afterwards to install a package and save it as a dependency in the package.json file.

Press `^C` at any time to quit.

package name: (servidor)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository:

keywords:

author:

license: (ISC)

About to write to `/Users/gian/UFFS/GEX613-ProgII/aulas/servidor/package.json`:

```
{
  "name": "servidor",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Is this OK? (yes) ☐

Criando um projeto Node.JS

yarn init

```
→ servidor yarn init
yarn init v1.22.19
question name (servidor):
question version (1.0.0):
question description:
question entry point (index.js):
question repository url:
question author:
question license (ISC):
question private:
success Saved package.json
🌟 Done in 8.97s.
```

package.json

```
{  
  "name": "servidor",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

npm init

```
{  
  "name": "servidor",  
  "version": "1.0.0",  
  "main": "index.js",  
  "license": "MIT"  
}
```

yarn init

package.json

```
npm init nome_do_modulo
```

ou

```
yarn add nome_do_modulo
```

```
yarn add express nodemon
```

```
{  
  "name": "servidor",  
  "version": "1.0.0",  
  "main": "index.js",  
  "license": "MIT",  
  "dependencies": {  
    "express": "^4.18.2"  
    "nodemon": "^3.0.1"  
  }  
}
```


index.js

```
import express from "express";
```

ou

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

```
const app = express();  
app.listen(3001, () => console.log("Servidor rodando na porta 3001"));
```

para iniciar o servidor:

```
nodemon index.js
```

GET request

```
app.get("/", (req, res) => {  
    res.send("Hello, world!");  
});
```

ou

```
app.get("/", function (req, res) {  
    res.send("Hello, world!");  
});
```

```
yarn add body-parser
```

```
{  
  "name": "servidor",  
  "version": "1.0.0",  
  "main": "index.js",  
  "license": "MIT",  
  "dependencies": {  
    "body-parser": "^1.20.2",  
    "express": "^4.18.2"  
  }  
}
```

POST request

```
const bodyParser = require('body-parser');
```

```
...
```

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

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

```
...
```

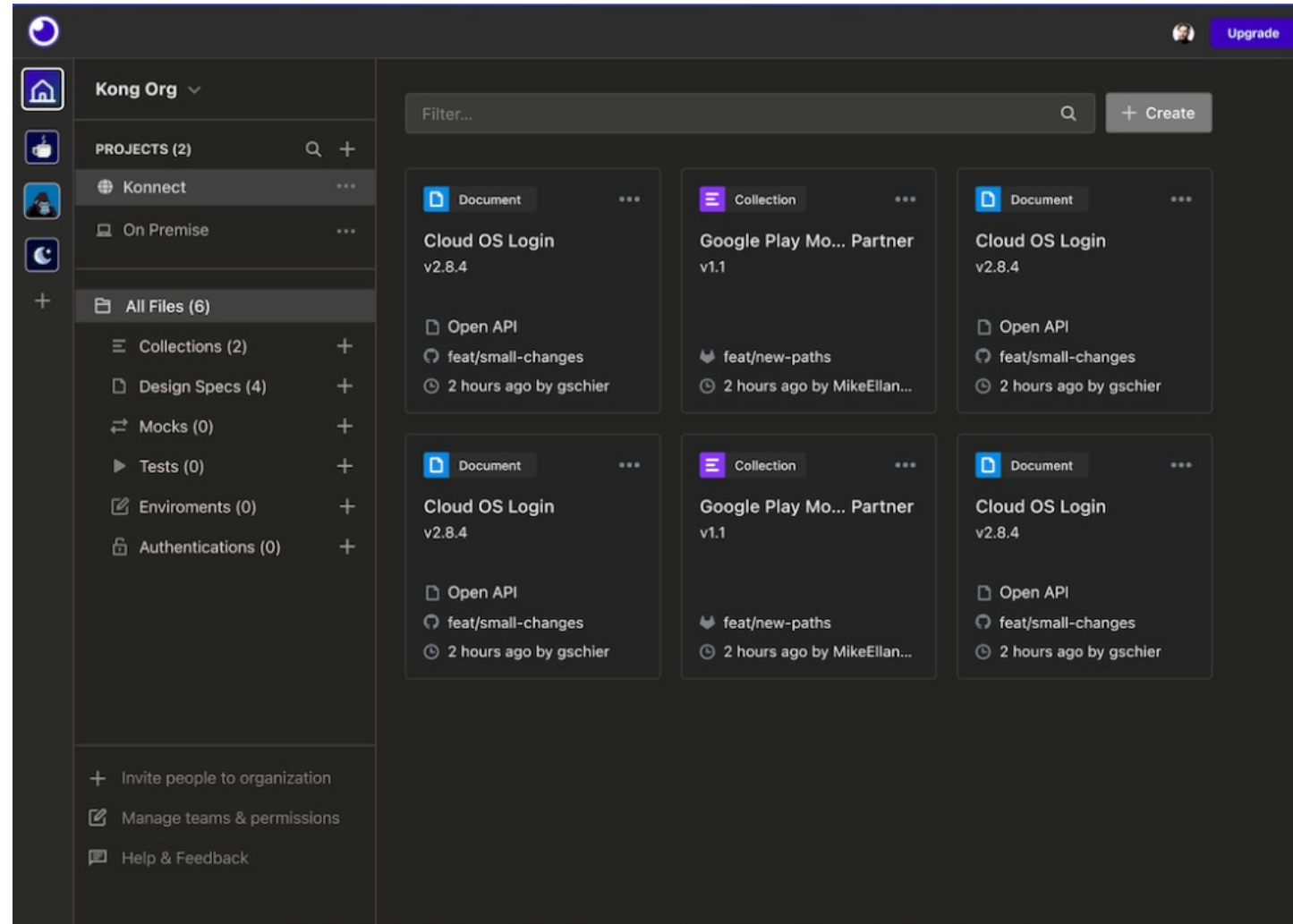
```
app.post("/", function (req, res) {
```

```
    const nome = req.body.nome;
```

```
    res.send(`Hello, ${nome}!`);
```

```
});
```

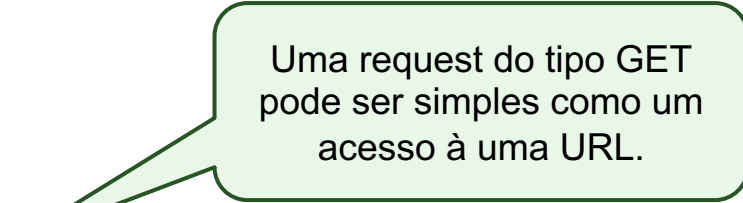
Insomnia REST



`http://localhost:3001/exemplo_get`

`http://localhost:3001/exemplo_query?nome_do_parametro=valor_do_parametro`

GET request



Uma request do tipo GET
pode ser simples como um
acesso à uma URL.

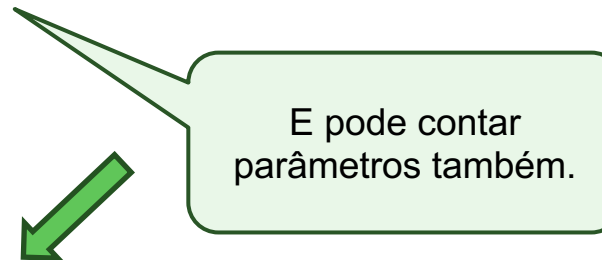
`http://localhost:3001/exemplo_get`

`http://localhost:3001/exemplo_query?nome_do_parametro=valor_do_parametro`

`http://localhost:3001/exemplo_get`

`http://localhost:3001/exemplo_query?nome_do_parametro=valor_do_parametro`

sintaxe:



`nome_do_parametro1=valor_do_parametro1&nome_do_parametro2=valor_do_parametro2`

POST request

`http://localhost:3001/exemplo_post`

`request body:`

```
{  
  nome_do_parametro1: "valor_do_parametro1",  
  nome_do_parametro2: "valor_do_parametro2"  
  ...  
}
```

POST request

`http://localhost:3001/exemplo_post`

Uma *request* do tipo POST serve para enviar valores para o servidor. Para isso, precisamos enviar os valores no corpo da *request*.

request body:

```
{  
  nome_do_parametro1: "valor_do_parametro1",  
  nome_do_parametro2: "valor_do_parametro2"  
  ...  
}
```

POST request

`http://localhost:3001/exemplo_post`

request body:

```
{  
  nome_do_parametro1: "valor_do_parametro1",  
  nome_do_parametro2: "valor_do_parametro2"  
  ...  
}
```

Observe que o corpo da *request* é um objeto JSON.