PROGRAMAÇÃO ORIENTADA A OBJETOS II

Profª Letícia Pieper Jandt

O que é uma API?

APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos.

Por exemplo, o sistema de software do instituto meteorológico contém dados meteorológicos diários. A aplicação para a previsão do tempo em seu telefone “fala” com esse sistema por meio de APIs e mostra atualizações meteorológicas diárias no telefone.

O que significa API?

API significa Application Programming Interface (Interface de Programação de Aplicação).

No contexto de APIs, a palavra Aplicação refere-se a qualquer software com uma função distinta.

O que significa API?

A interface pode ser pensada como um contrato de serviço entre duas aplicações. Esse contrato define como as duas se comunicam usando solicitações e respostas.

A documentação de suas respectivas APIs contém informações sobre como os desenvolvedores devem estruturar essas solicitações e respostas.

Como funciona uma API?

A arquitetura da API geralmente é explicada em termos de cliente e servidor.

A aplicação que envia a solicitação é chamada de cliente e a aplicação que envia a resposta é chamada de servidor.

Como funciona uma API?

Então, no exemplo do clima, o banco de dados meteorológico do instituto é o servidor e o aplicativo móvel é o cliente.

Existem quatro maneiras diferentes pelas quais as APIs podem funcionar, dependendo de quando e por que elas foram criadas.

Como funciona uma API?

APIs REST

Essas são as APIs mais populares e flexíveis encontradas na Web atualmente.

O cliente envia solicitações ao servidor como dados. O servidor usa essa entrada do cliente para iniciar funções internas e retorna os dados de saída ao cliente.

Api rest

REST significa Transferência Representacional de Estado.

REST define um conjunto de funções como GET, PUT, DELETE e assim por diante, que os clientes podem usar para acessar os dados do servidor. Clientes e servidores trocam dados usando HTTP.

Api rest

A principal característica da API REST é a ausência de estado.

A ausência de estado significa que os servidores não salvam dados do cliente entre as solicitações. As solicitações do cliente ao servidor são semelhantes aos URLs que você digita no navegador para visitar um site.

A resposta do servidor corresponde a dados simples, sem a renderização gráfica típica de uma página da Web.

Node.JS: o que é, como funciona

O Node.js por sua vez, é uma plataforma open source que permite a execução de código JavaScript a nível front-end e back-end.

Em palavras mais simples, o Node.js é uma forma de executar o JavaScript do lado do servidor de uma aplicação.

Node.JS: o que é, como funciona

Teve seu lançamento em 2009 sobre a licença MIT e é utilizado por diversas grandes empresas como LinkedIn, Groupon, PayPal, entre outras.

O Node.js possui aplicabilidade em diversos meios, dentre eles podemos citar:

- Criação de aplicações de chats e mensagens instantâneas;
- Criação de API´s escaláveis;
- Aplicações web que funcionam em real-time;
- Aplicações CLI (Client Line Interface), entre outros.

Node.JS: o que é, como funciona

Com a evolução das tecnologias web, tornou-se possível fazer o JavaScript rodar também no Back-end, e é nesse momento de consolidação de tecnologias e soluções que surge o Node.js.

Node.JS: o que é, como funciona

O Node.js é um ambiente de execução do código JavaScript do lado servidor (server side), que na prática se reflete na possibilidade de criar aplicações standalone (autossuficientes) em uma máquina servidora, sem a necessidade do navegador.

Node.JS: o que é, como funciona

O Node.js é a ferramenta que vai nos entregar a capacidade de interpretar código JavaScript, de maneira bem similar ao navegador. Quando executamos um comando escrito em JavaScript, o Node.js interpreta esse comando e faz a sua conversão para a linguagem de máquina a ser executada pelo computador.

O que é express.js?

Tendo sua versão inicial lançada no ano de 2010, o Express.js (ou somente Express) é um Framework para o desenvolvimento de aplicações JavaScript com o Node.js.

De código aberto, sobre a licença MIT, o Express.js foi desenvolvido para otimizar a construção de aplicações web e APIs.

O que é express.js?

O Express.js é um Framework rápido e um dos mais utilizados em conjunto com o Node.js, facilitando no desenvolvimento de aplicações back-end e até, em conjunto com sistemas de templates, aplicações full-stack.

Características express.js

principais características do express:

- Possui um sistema de rotas completo;
- Possibilita o tratamento de exceções dentro da aplicação;
- Permite a integração de vários sistemas de templates que facilitam a criação de páginas web para suas aplicações;
- Gerencia diferentes requisições HTTP com seus mais diversos verbos;
- Feito para a criação rápida de aplicações utilizando um conjunto pequeno de arquivos e pastas;

MVC

O que é MVC? (Model – View – Controller)

O MVC é um padrão de arquitetura de software que separa a sua aplicação em 3 camadas.

O princípio básico do MVC é a divisão da aplicação em três camadas: a camada de interação do usuário (view), a camada de manipulação dos dados (model) e a camada de controle (controller).

MVC

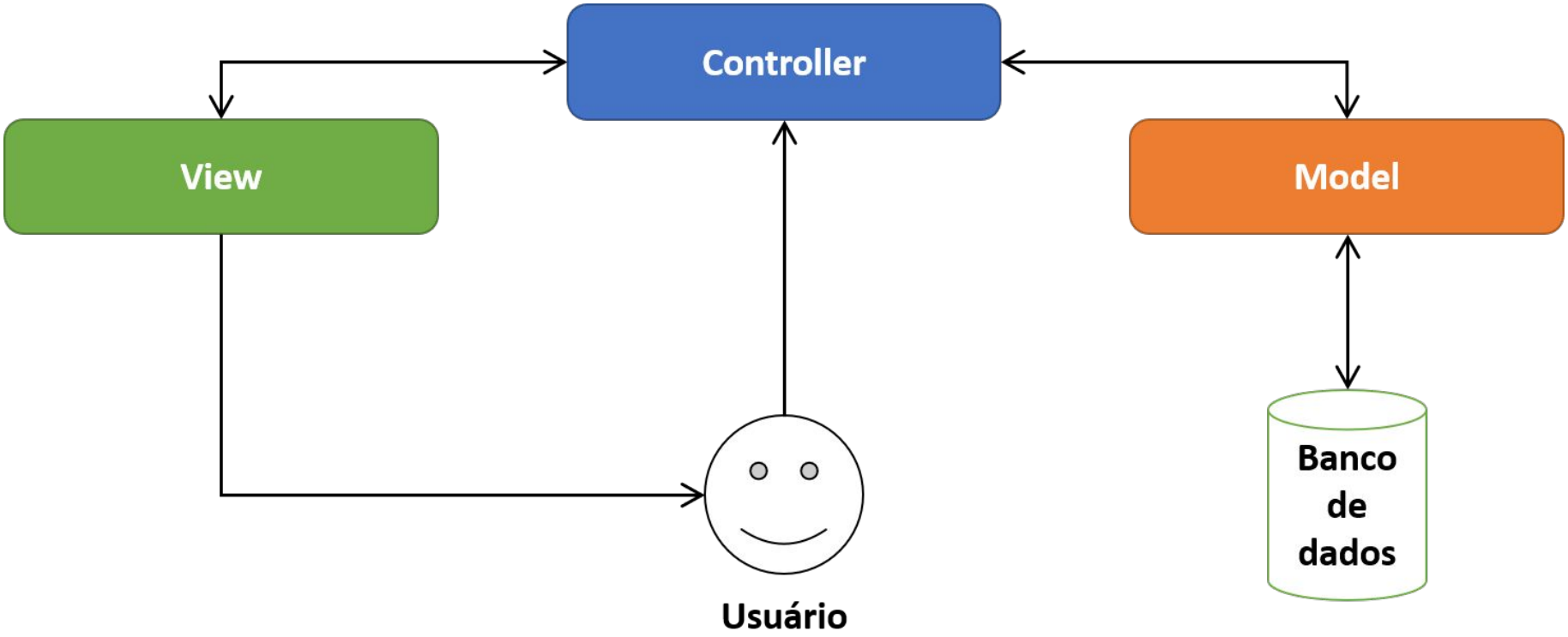
O MVC é um padrão de arquitetura de software. O MVC sugere uma maneira para você pensar na divisão de responsabilidades, principalmente dentro de um software web.

Quais os papéis de cada camada?

Quando falamos sobre o MVC, cada uma das camadas apresenta geralmente as seguintes responsabilidades:

- Model - A responsabilidade dos models é representar o negócio. Também é responsável pelo acesso e manipulação dos dados na sua aplicação.
- View - A view é responsável pela interface que será apresentada, mostrando as informações do model para o usuário.
- Controller - É a camada de controle, responsável por ligar o model e a view, fazendo com que os models possam ser repassados para as views e vice-versa.

Quais os papéis de cada camada?



O MVC e sua importância

Uma dessas vantagens é que ele nos ajuda a deixar o código mais manutenível, ou seja, mais fácil de fazer manutenção, já que temos as responsabilidades devidamente separadas.

Isso também traz uma facilidade na compreensão do código, além da sua reutilização.

O MVC e sua importância

Além disso, você tem um código mais testável.

se você tem uma aplicação onde, por exemplo, na página de listagem de usuários, o nome do usuário está sendo cortado ou não está sendo exibido da maneira correta, é muito mais fácil você fazer um teste que atinja somente as estruturas de views.

O que são controllers?

O Controller é uma terça parte do padrão de arquitetura chamado "MVC" (Model-View-Controller).

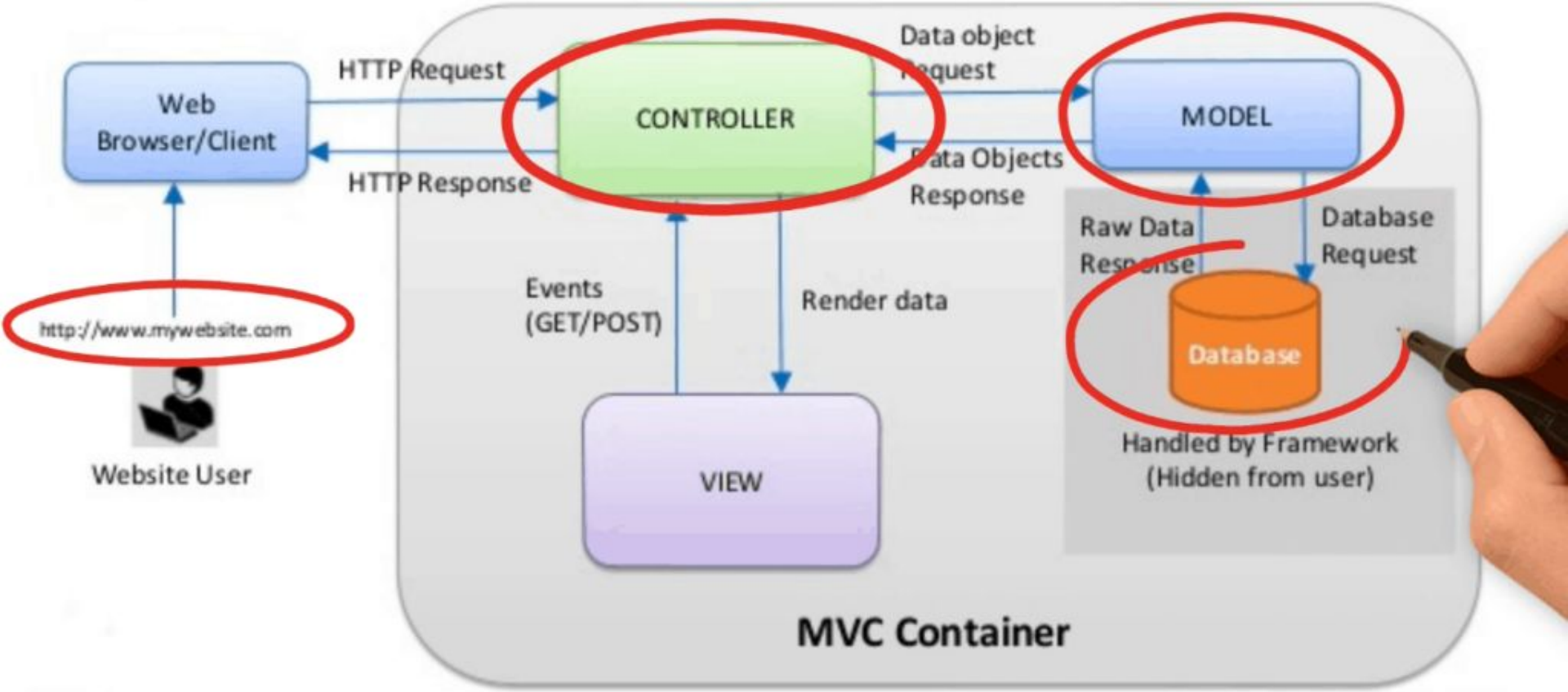
O Controller é responsável por receber todas as requisições do usuário. Seus métodos chamados actions são responsáveis por uma página, controlando qual model usar e qual view será mostrado ao usuário.

O que são controllers?

Com os controllers, também é possível gerenciar os verbos HTTP para cada método da classe, permitindo criar todos os acessos RestFul para sua api.

por eles fazemos os mapeamentos e endpoints

Deixando o código mais limpo e organizado.


Web Browser/Client
HTTP Request
HTTP Response
CONTROLLER
Data object Request
Data Objects Response
MODEL
Raw Data Response
Database Request
Database
Handled by Framework (Hidden from user)
Events (GET/POST)
Render data
VIEW
http://www.mywebsite.com
Website User
MVC Container

Vamos começar nossa api

O primeiro passo é instalar o Node em sua máquina

depois - abra o cmd como administrador

e execute o comando:

npm install -g express-generator

construindo - estrutura

Vamos criar um diretório (um projeto)

criar uma pasta (nome sem acento, sem caracteres especiais...)

entrar na pasta com o cmd

e executar o comando

npm init -y

Esse comando é para gerar o arquivo package.json que vai servir como base de configuração da nossa aplicação

construindo - server.js

O arquivo server.js vai ser usado para a inicialização do nosso projeto.

crie uma **pasta chamada bin** e **dentro dela** o arquivo **server.js**

```
const app = require('../src/app');
const port = normalizaPort(process.env.PORT || '3000');

function normalizaPort(val) {
    const port = parseInt(val, 10);
    if (isNaN(port)) {
        return val;
    }

    if (port >= 0) {
        return port;
    }

    return false;
}

app.listen(port, function () {
    console.log(`app listening on port ${port}`)
})
```

construindo - server.js

nós estamos importando um módulo que iremos criar nos próximos passos. Depois, estamos definindo uma porta para que ele seja executado, no final estamos passando para o método app.listen a porta que queremos que ele escute o nosso projeto e de um console.log com ela.

Controllers

Para que possamos organizar o nosso código, nós dividimos ele pensando em um padrão MVC.

Crie a pasta src e **dentro dela a pasta controllers** e ai **crie o arquivo PessoaController.js**

O MVC sugere uma maneira para você pensar na divisão de responsabilidades, principalmente dentro de um software web. O princípio básico do MVC é a divisão da aplicação em três camadas: a camada de interação do usuário (view), a camada de manipulação dos dados (model) e a camada de controle (controller).

```
exports.get = (req, res, next) => {
    res.status(200).send('Requisição recebida com sucesso!');
};

exports.getById = (req, res, next) => {
    res.status(200).send('Requisição recebida com sucesso!');
};

exports.post = (req, res, next) => {
    res.status(201).send('Requisição recebida com sucesso!');
};

exports.put = (req, res, next) => {
    let id = req.params.id;
    res.status(201).send(`Requisição recebida com sucesso! ${id}`);
};

exports.delete = (req, res, next) => {
    let id = req.params.id;
    res.status(200).send(`Requisição recebida com sucesso! ${id}`);
};
```

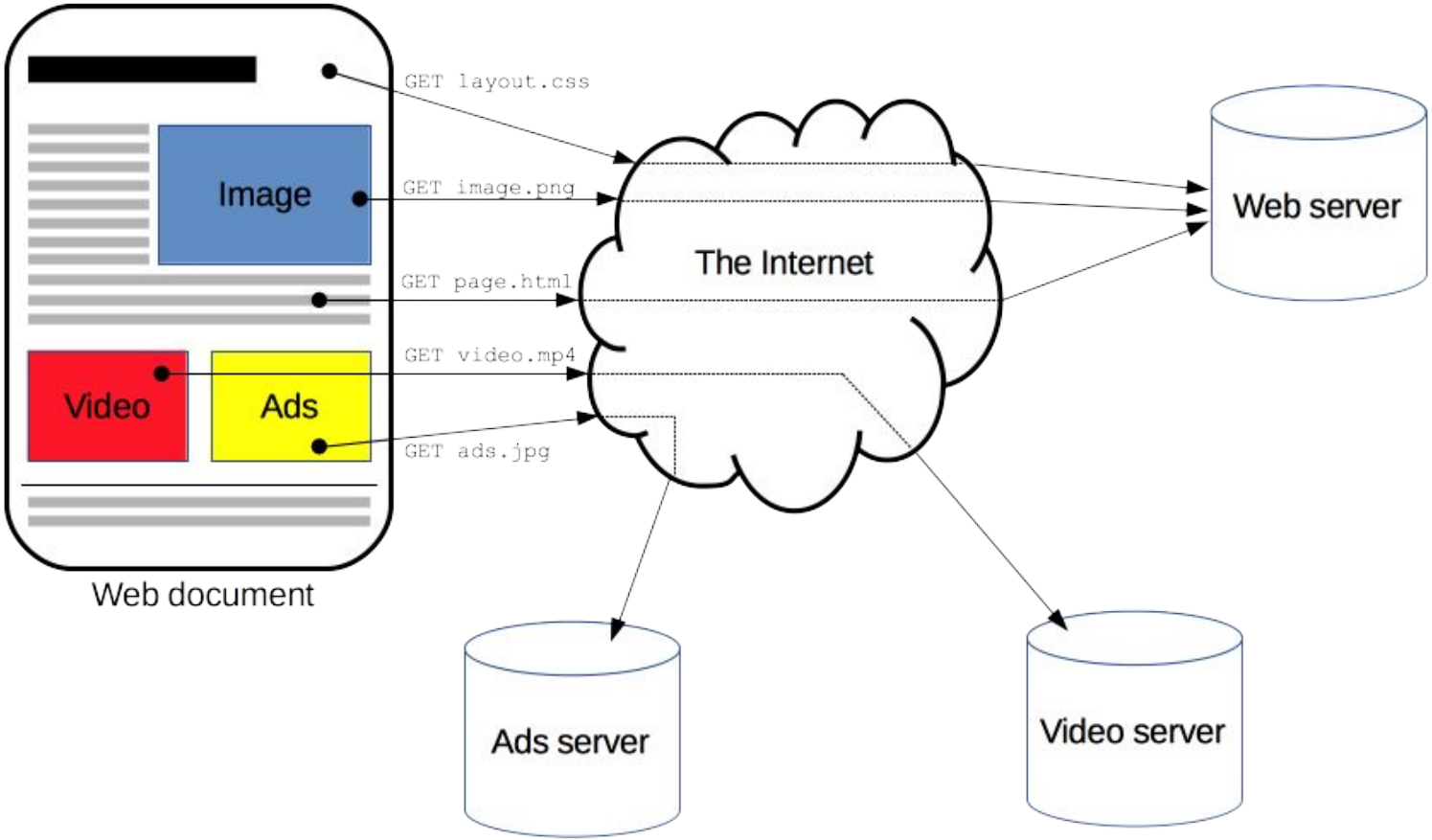
PessoaController

Protocolo HTTP

HTTP é um protocolo que permite a obtenção de recursos, tais como documentos HTML.

É a base de qualquer troca de dados na Web e um protocolo cliente-servidor, o que significa que as requisições são iniciadas pelo destinatário, geralmente um navegador da Web.

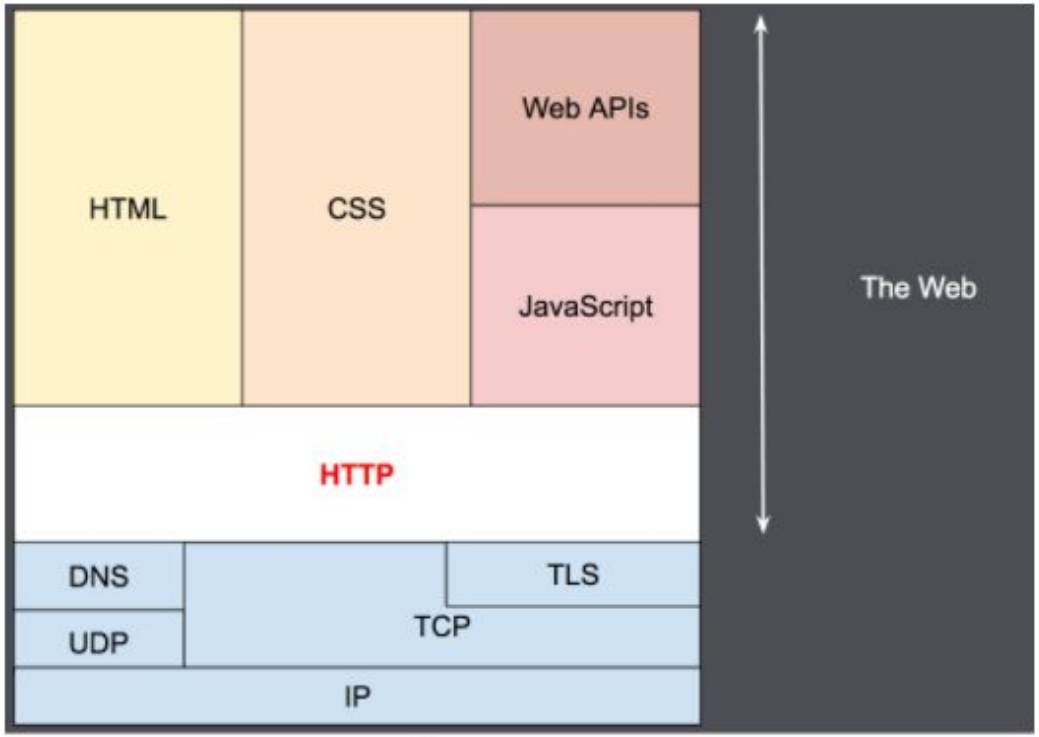
Protocolo HTTP



Protocolo HTTP

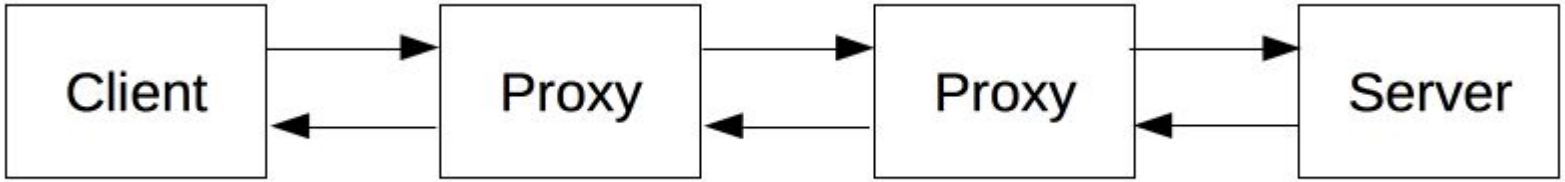
Clientes e servidores se comunicam trocando mensagens individuais (em oposição a um fluxo de dados).

As mensagens enviadas pelo cliente, geralmente um navegador da Web, são chamadas de solicitações (requests), ou também requisições, e as mensagens enviadas pelo servidor como resposta são chamadas de respostas (responses).



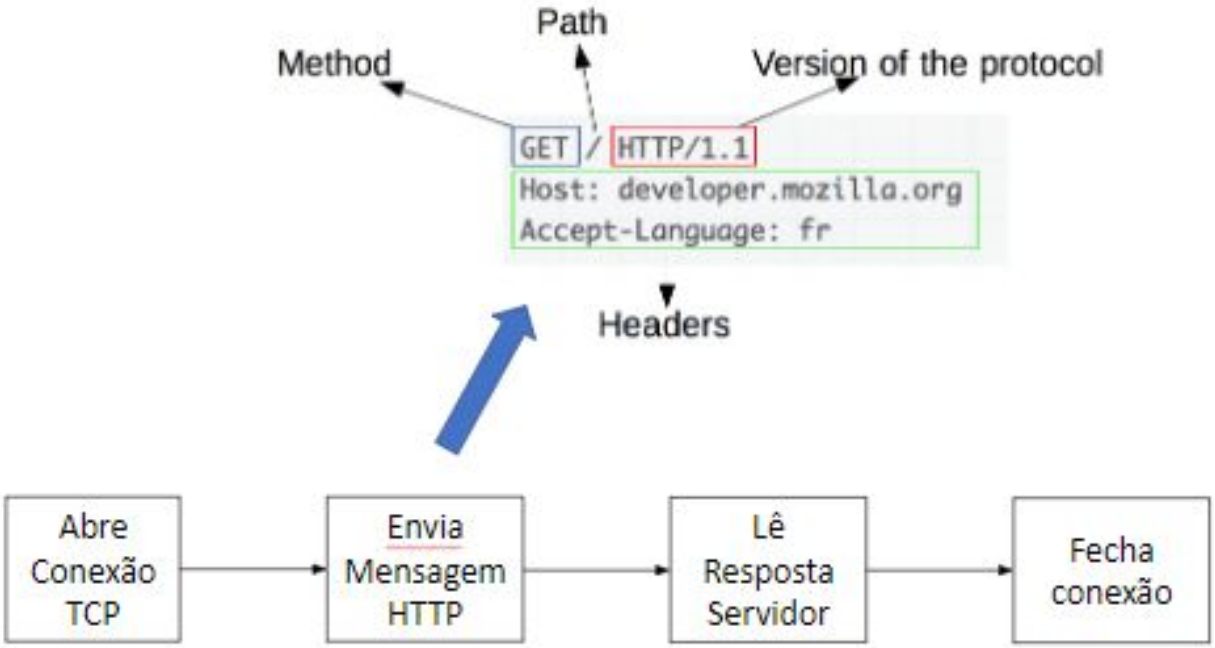
Projetado no início da década de 1990, o HTTP é um protocolo extensível que evoluiu ao longo do tempo. É um protocolo de camada de aplicação que é enviado sobre TCP, ou em uma conexão TCP criptografada com TLS, embora qualquer protocolo de transporte confiável possa, teoricamente, ser usado. Devido à sua extensibilidade, ele é usado para não apenas buscar documentos de hipertexto, mas também imagens e vídeos ou publicar conteúdo em servidores, como nos resultados de formulário HTML (veja os elementos `<html>` e `<form>`). O HTTP também pode ser usado para buscar partes de documentos para atualizar páginas da Web sob demanda.

Request - Response



Entre a solicitação e a resposta existem várias entidades, designadas coletivamente como proxies, que executam operações diferentes e atuam
como *gateways* (intermediários) ou caches, por exemplo

Fluxo HTTP


Path
Method
Version of the protocol
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
Headers
Abre Conexão TCP
Envia Mensagem HTTP
Lê Resposta Servidor
Fecha conexão

Métodos HTTP

GET

POST

PUT

DELETE

HTTP Responses

200 – OK
400 – Bad Request
404 – Not Found
405 – Method Not Allowed

Rotas

As rotas servem para direcionar os usuários para as ações que eles querem realizar.

Nessa parte, nós temos dois arquivos: index.js e PessoaRouter.js.

O arquivo index.js seria para passar a versão que está a nossa API ou para que possamos passar para um balanceador (Load Balancer) verificar se a nossa API está no ar.

O PessoaRouter.js contém as rotas que iremos utilizar para nossa PessoaController

dentro de src crie a pasta routes e o arquivo index.js

src/routes/index.js

```
1   const express = require('express');
2   const router = express.Router();
3
4   router.get('/', function (req, res, next) {
5       res.status(200).send({
6           title: "Minha primeira api",
7           version: "0.0.1"
8       });
9   });
10  module.exports = router;
```

src/routes/PessoaRouter.js

```
1   const express = require('express');
2   const router = express.Router();
3   const controller = require('../controllers/PessoaController')
4
5   router.get('/', controller.get);
6   router.get('/:id', controller.getById);
7   router.post('/', controller.post);
8   router.put('/:id', controller.put);
9   router.delete('/:id', controller.delete);
10
11  module.exports = router;
```

Configurações

O arquivo app.js é responsável pelas configurações do nosso projeto. É nele que nós devemos configurar a nossa base de dados, rotas etc.

crie o app.js na raiz do projeto.

src/app.js

src > JS app.js > ...

```
1   const express = require('express');
2   const app = express();
3
4   //Rotas
5   const index = require('./routes/index');
6   const pessoaRoute = require('./routes/PessoaRouter');
7
8   app.use(express.json())
9   app.use(express.urlencoded({ extended: true }))
10
11  app.use('/', index);
12  app.use('/pessoa', pessoaRoute);
13
14  module.exports = app;
15
```

Nodemon

O pacote nodemon nos auxilia no momento do nosso desenvolvimento. Com ele, nós não precisamos dar stop e subir novamente a nossa APP. Ele verifica que ocorreu uma alteração e já faz o refresh automaticamente.

Para instalá-lo, execute o comando na sua console (dentro do cmd do seu projeto):

```
npm install -g nodemon
```

Arquivo Package.config

Esse seria o arquivo inicial nos projetos Node, nele nós temos todas as dependências:

Testes

Para que possamos testar o nosso projeto, **digite o comando npm install na sua console** para importar os pacotes necessários para a nossa aplicação.

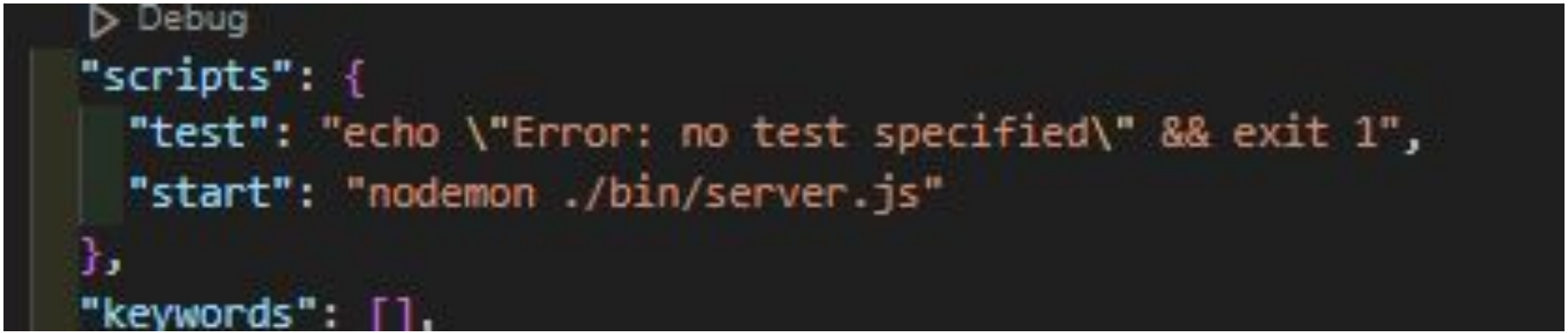
```
C:\Users\letic\Documents\PESSOAL\PIEPER04\FASIPE\POO II\2024\AULA 002\poo2>npm install

up to date, audited 1 package in 414ms

found 0 vulnerabilities
```

Testes

vamos adicionar o script de start no arquivo package.json



Testes

agora no cmd, execute o comando npm start. Caso tudo esteja ok nos passos anteriores, você irá ver a mensagem abaixo na sua console:

```
C:\Users\letic\Documents\PESSOAL\PIEPER04\FASIPE\POO II\2024\AULA 002\poo2>npm start

> poo2@1.0.0 start
> nodemon ./bin/server.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./bin/server.js`
app listening on port 3000
```

ação

Agora, abra no seu navegador o endereço http://localhost:3000/.

http://localhost:3000/pessoa

http://localhost:3000/pessoa/1

Exercícios

https://www.treinaweb.com.br/blog/o-que-e-mvc