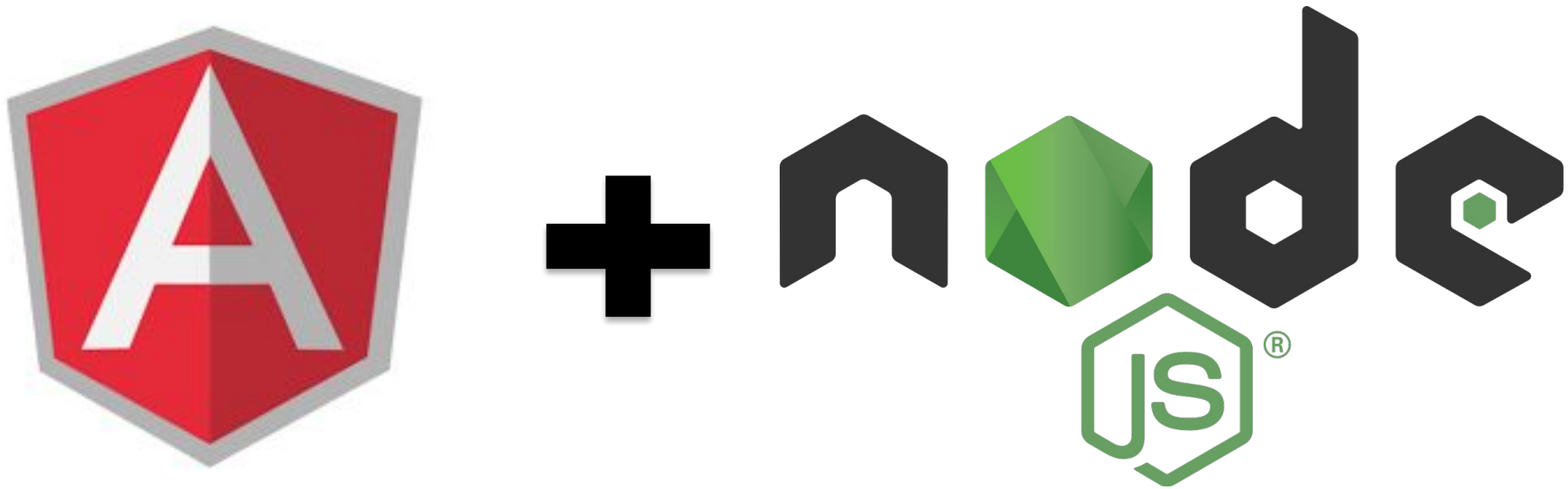

node
JS

Uma filosofia...Mean

Mean é a iniciativa de usar apenas ferramentas feitas em Javascript para desenvolvimento end-end.

Mean significa:

Mongo = Banco de Dados NoSQL
Express = Framework de NodeJS
Angular = Framework de Javascript para frontend
Node.JS = Servidor Backend em JavaScript

Como começar a utilizar o Mean?

1 – Baixe e instale o Mongo.

2 – Baixe e instale o Node.JS

3 – Pelo Node Package Manager(npm) instale o Angular CLI.

4 – Ao começar um projeto de Back-End(Node.JS) use o npm para baixar o framework express.

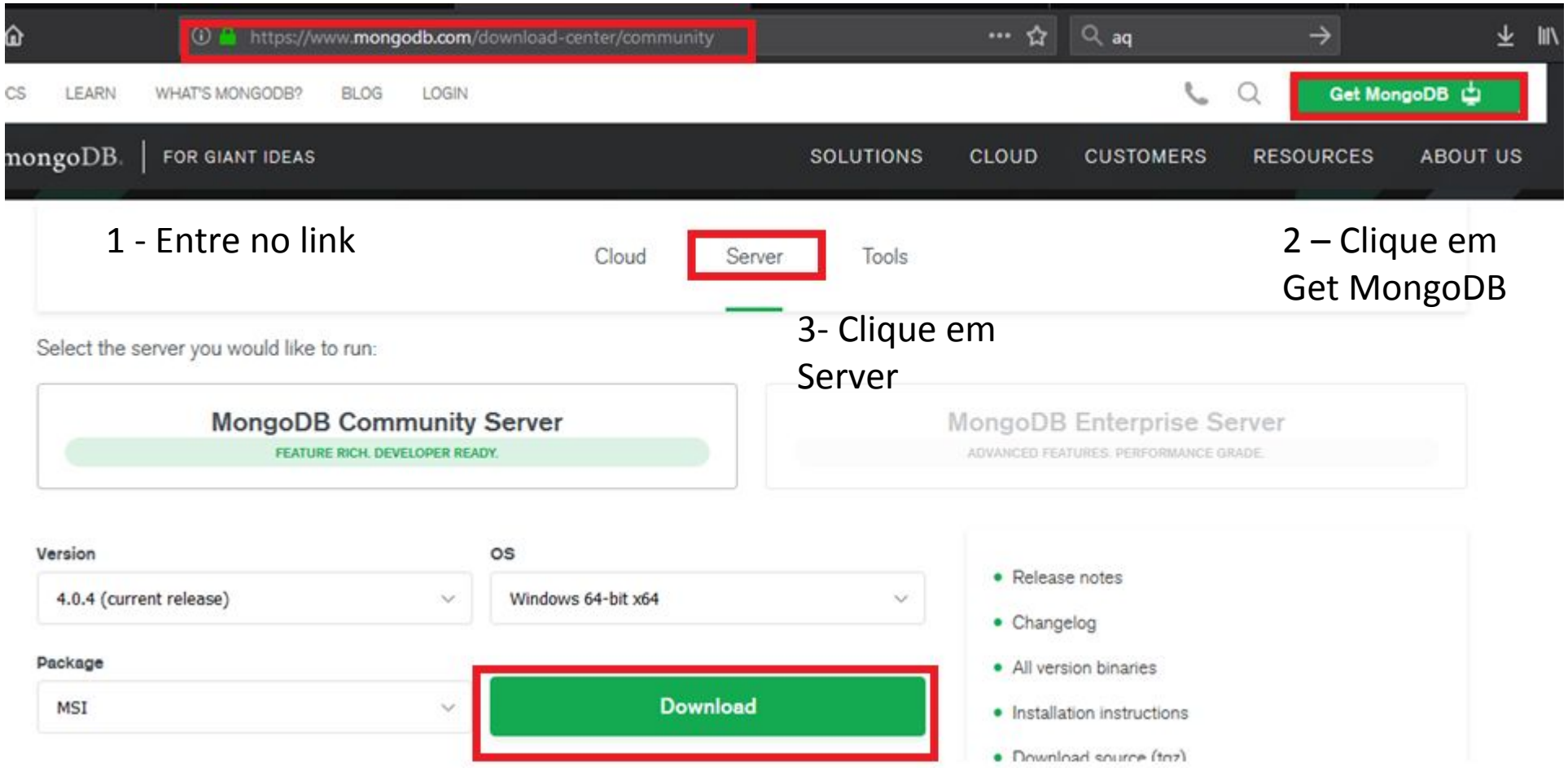
Baixar e instalar o Mongo(Ubuntu)

Digite esse comando no terminal do Ubuntu:

```
sudo apt-get install -y mongodb-org
```

Depois apenas digite “mongo” para abrir o banco de dados.

Baixar e instalar o Mongo(Windows)



1 - Entre no link

2 – Clique em Get MongoDB

3- Clique em Server

4 – Escolha a versão e clique em download

5 – Depois só executar e instalar.

Baixar e instalar o Node.JS(Ubuntu)

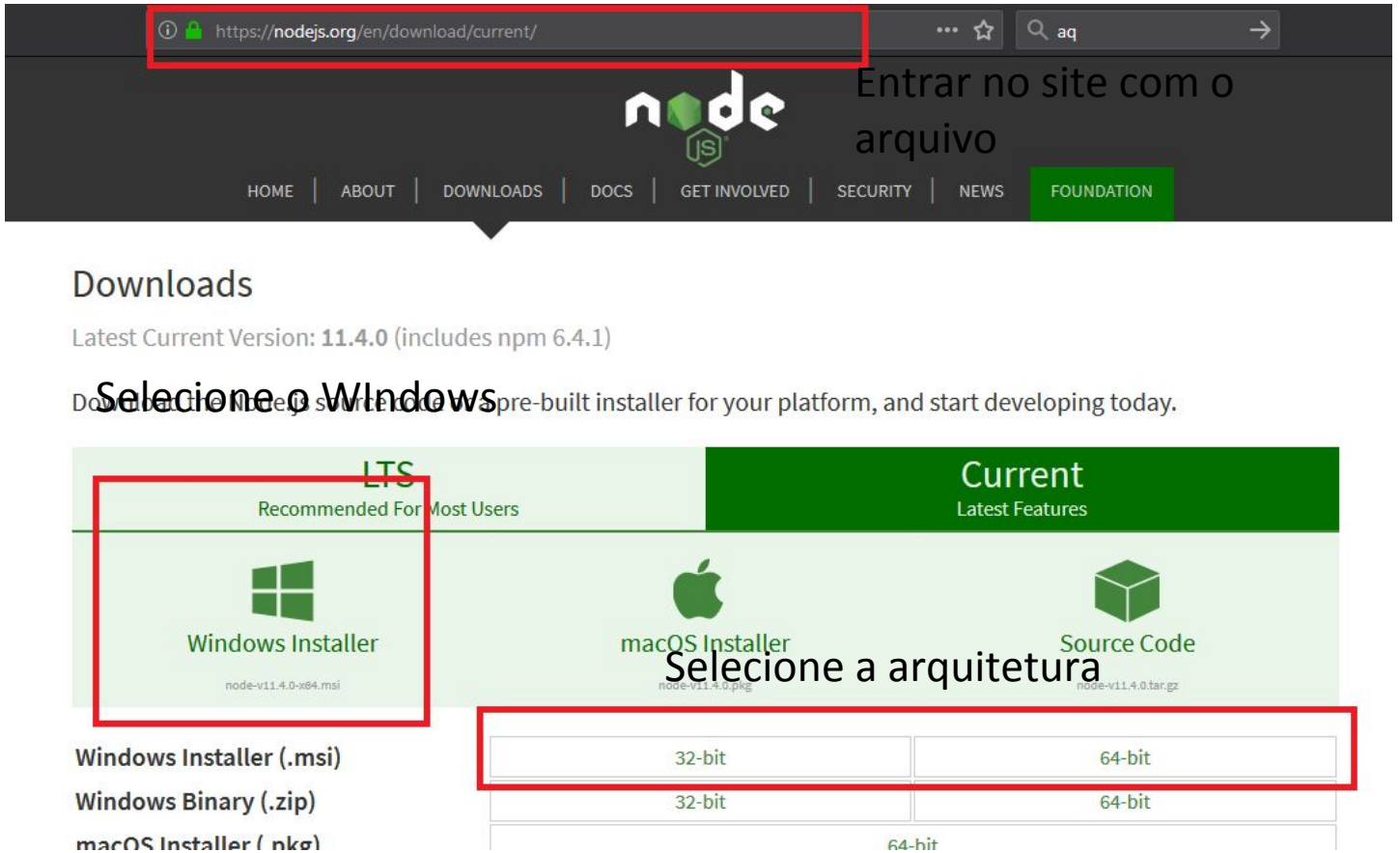
Digite os seguintes comandos no terminal do Ubuntu:

```
sudo apt install npm
```

```
sudo apt install nodejs
```

Depois apenas digite “node –version” para confirmar se está instalado.

Baixar e instalar o Node.JS(Windows)



Se estiver usando Windows, garanta que a pasta que contenha o Node e o NPM estejam no path, da mesma forma o MongoDB.

Instalação Completada

Agora vamos ao Node.JS...

Node.JS e Express

O que é o Node.JS?

Interpretador de código Javascript focado em servidores.

Ele que vai ser a conexão entre o front-end e o banco de dados.

Entretanto, trabalhar com Node puro pode gastar muito tempo para executar tarefas simples, por isso ele tem uma coletânea de frameworks para facilitar o processo.

O mais conhecido é o Express.

Como trabalhar com o Node?

Facilitará muito a vida do programador se ele utilizar o terminal.

Crie um diretório e entre nele, para começar um projeto se deve usar o npm e iniciar um projeto digitando “npm init”, ele vai criar uma pasta e um arquivo chamado package.json que vai gerenciar os pacotes que forem instalados futuramente.

Quais frameworks vamos usar com o NodeJS?

1 – express, que é o que vai facilitar o controle de requisições http.

2 – body-parser, vai permitir que envie respostas JSON prontas como páginas de html.

3 – cors, vai permitir quer um servidor externo acesso os dados do NodeJS, que por segurança não permite por padrão.

4 - mongoose, o jeito mais fácil de acessar dados do MongoDB.

5 – morgan, um middleware para gerar logs de requisições.

Comandos para instalar os Frameworks

1 – Express: “npm install express --save”

2 – BodyParser: “npm install body-parser --save”

3 – Cors: “npm install cors --save”

4 - Mongoose:”npm install mongoose --save”.

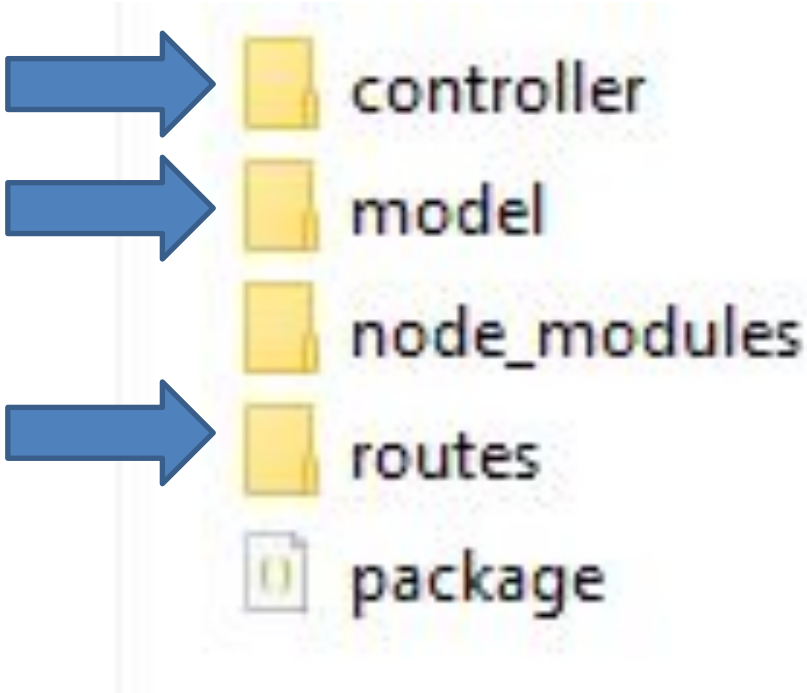
5 – Morgan: “npm install Morgan --save”

Depois de tudo pronto.

Vamos criar o arquivo para conectar ao Mongo.

Parte 1

Criar as pastas:



Agora....

Depois de criar a estrutura dos diretórios, é hora de criar o modelo de documento para salvar no MongoDB.

Parte 2

Na pasta model, criar o arquivo usuario-model.js

```
'use strict';

const mongoose = require('mongoose');
mongoose.set('useCreateIndex', true);
const Schema = mongoose.Schema;

const schema = new Schema({
    nome:{
        type: String,
        required: true,
        unique: true
    },
    senha:{
        type: String,
        required: true
    }
})

module.exports = mongoose.model('Usuario', schema, 'usuario');
```

Agora....

Com o nosso model de usuário criado, implementaremos o código responsável pelas operações CRUD.

Parte 3

Na pasta controller, criar o arquivo usuário-controller.js

```
const mongoose = require('mongoose');
const Usuario = mongoose.model('Usuario');

exports.get = (req, res, next) => {
    Usuario.find(function (err, docs) {
        res.send(docs);
    });
}

exports.post = (req, res, next) => {
    var usuario = new Usuario(req.body);
    usuario.save(function (err) {
        if (err) return handleError(err);
    });
    res.send({resposta:'salvo com sucesso'});
}
```

Parte 5

Ainda no arquivo usuario-controller.js ...

```
exports.put = (req, res, next) => {
    Usuario.findByIdAndUpdate(req.params.id, req.body, (err, todo) => {
        if (err) return res.status(500).send(err);
        return res.send({resposta:'atualizado com sucesso'});
    })
}

exports.delete = (req, res, next) => {
    Usuario.findByIdAndDelete(req.params.id, (err, todo) => {
        if (err) return res.status(500).send(err);
        return res.send({resposta:'deletado com sucesso'});
    })
}
```

Agora....

Com as ações de CRUD criadas, definiremos como elas serão acessadas...

Parte 6

Na pasta routes, crie o arquivo usuarios-route.js

```
'use strict'

const express = require('express');
const router = express.Router();
const Usuario = require('../model/usuario-model')

const controller = require('../controller/usuarios-controller');

router.get('/usuarios', controller.get);

router.post('/usuarios', controller.post);

router.put('/usuarios/:id', controller.put);

router.delete('/usuarios/:id', controller.delete);

module.exports = router;
```

Agora....

Com as rotas prontas, agora iremos implementar o código para iniciar o servidor node.

Parte 7

Na pasta routes, crie o arquivo usuarios-route.js

```
'use strict'

const express = require('express');
const router = express.Router();
const Usuario = require('../model/usuario-model')

const controller = require('../controller/usuarios-controller');

router.get('/usuarios', controller.get);

router.post('/usuarios', controller.post);

router.put('/usuarios/:id', controller.put);

router.delete('/usuarios/:id', controller.delete);

module.exports = router;
```

Agora....

Com as rotas prontas, agora iremos implementar o código para iniciar o servidor node.

Parte 8

Na pasta raiz do api, crie o arquivo app.js

```
'use strict'
const express = require('express');
const app = express();
const cors = require('cors');
const bodyParser = require('body-parser')
const mongoose = require('mongoose');
const usuariosRoute = require('./routes/usuarios-route');
const morgan = require('morgan');
```

Parte 8

Ainda no arquivo app.js

```
mongoose.connect('mongodb://127.0.0.1:27017/web', { useNewUrlParser: true });

app.use(morgan('dev'));

app.use(bodyParser.json())

app.use(cors())

app.use('/', usuariosRoute);

app.listen(3000);
```

Agora....

Rode o backend utilizando a síntaxe “npm start” no terminal e pronto, seu servidor back-end está pronto, teste com simulador de requisições se quiser.

Agora é hora do front-end.

Angular

- Angular é uma plataforma e framework OpenSource desenvolvida pela Google para a criação de aplicativos web do lado usando HTML, CSS e TypeScript com foco no desenvolvimento de *Single-Page Applications(SPA)*.

Single-Page Applications

- A aplicação estará quase toda no cliente, sendo que assim que o usuário acesso o site a aplicação e seus templates são armazenados no lado cliente.

Blocos Principais

Angular CLI

- O Angular CLI é uma ferramenta de interface de linha de comando usada para inicializar, desenvolver, estruturar e manter aplicativos Angular.

Instalar o Angular CLI(Ambos SO)

Apenas digite no terminal:

```
npm install -g @angular/cli
```

Criar um Angular Workspace

Um espaço de trabalho contém os arquivos para um ou mais projetos. Um projeto é o conjunto de arquivos que compõe um aplicativo, uma biblioteca ou testes de ponta a ponta (e2e).

Comando:

```
ng new front-end
```

Angular CLI, comandos mais usados

ng new “nome da aplicação”

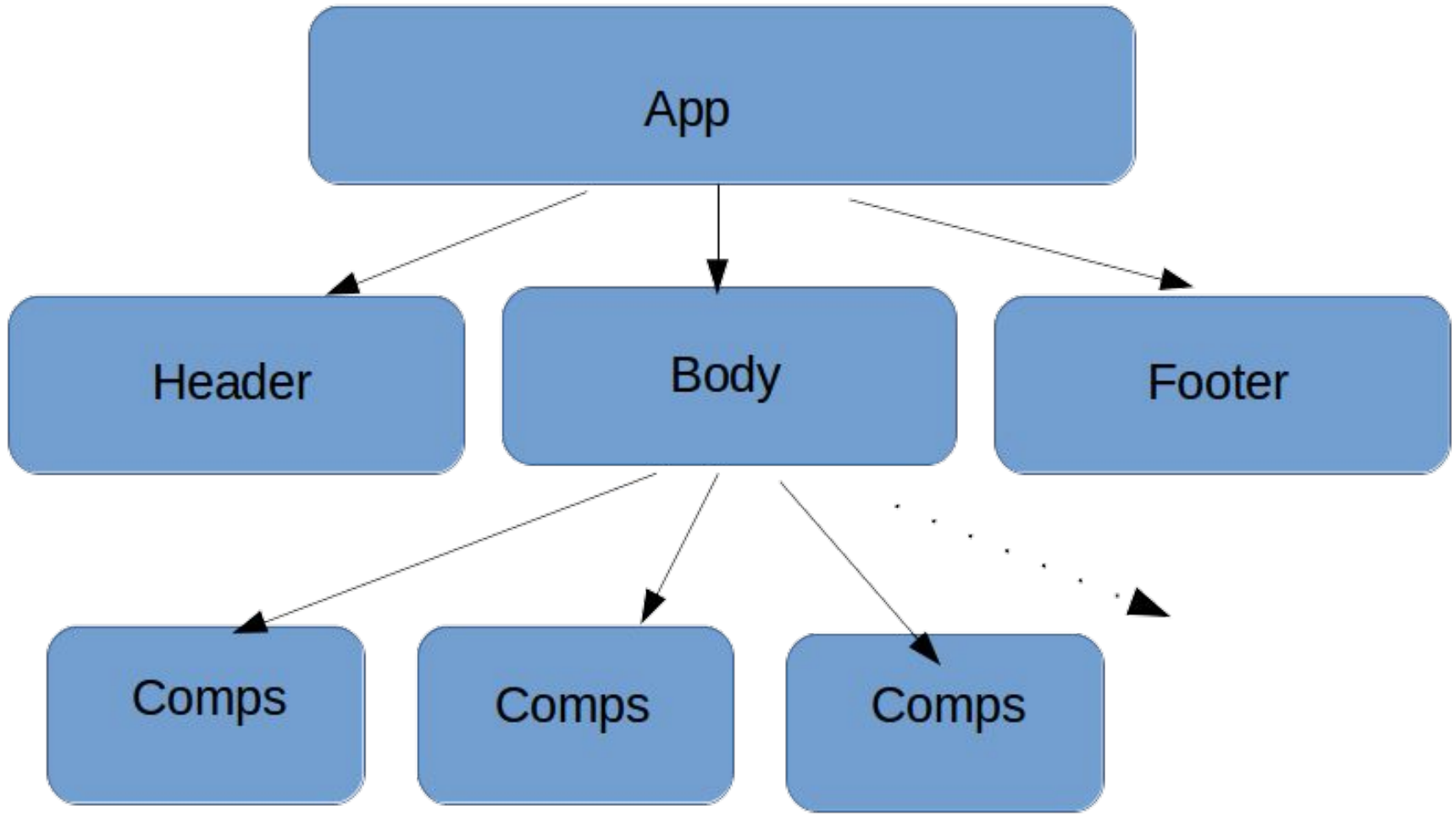
ng generate component “nome do component”

ng generate service “nome do serviço”

ng test (Executa testes automatizados)

ng serve (Inicia um servidor e abre a aplicação)

Componentes



Componentes

Componentes

Encapsula:

- Template
- Metadata: processamento das classes
- Data Binding
- Comportamento da View

```
ng generate component listarUsuario
```

Diretivas

- Responsável por modificar os elementos DOM e o seu comportamento

```
<ul>
  <li *ngFor=" let usuario of usuarios">
    <span class="text">Usuario: {{usuario.nome}}    Senha: {{usuario.senha}}</span>
  </li>
</ul>
```

Serviço(Service)

- Um serviço é tipicamente uma classe com um propósito estreito e bem definido.
- Para dados ou lógica que não estão associados a uma View específica e que você deseja compartilhar entre componentes, crie uma classe de serviço.

Injeção de Dependência

- Componentes consomem serviços, ou seja, você pode injetar um serviço em um componente, dando acesso ao componente para essa classe de serviço.

Two way data binding

O dados fluem nas duas direções: do html para o componente.ts e vice e versa.

Router

- O Angular Router NgModule fornece um serviço que permite definir um caminho de navegação entre os diferentes estados do aplicativo e exibir hierarquias no seu aplicativo.

Iniciando o projeto front-end

Clone ou baixe o projeto disponivel em:
https://github.com/gillallifam/DES_WEB

Abra o cmd e acesse a pasta front-end dentro da pasta do projeto.

Execute o comando npm install para installar todas as dependências.

Após isso abra a pasta front-end no editor (recomendamos o VsCode).

Criando um component

Execute o comando ng generate component “cadusuario”. O comando pode ser abreviado para “ng g c ‘nome do component’”.

Abra o arquivo app/cadusuario/cadusuario.component.ts

Apague o conteúdo e copie o código abaixo.

```
import { Component, OnInit } from '@angular/core';
import { HttpService } from '../http.service';
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';

@Component({
  selector: 'app-cadusuario',
  templateUrl: './cadusuario.component.html',
  styleUrls: ['./cadusuario.component.css']
})
export class CadusuarioComponent implements OnInit {
  usuario = {
    "nome" : "",
    "senha" : ""
  }

  constructor(public httpService : HttpService, private route: ActivatedRoute,  private location: Location) {
    this.httpService.location = location;
  }

  ngOnInit() {}
  cadastrar() {
    this.httpService.cadastrarUsuario(this.usuario);
  }
}
```

Faça o mesmo no arquivo cadusuario.component.html.

```
<h1>Nome: </h1>
<input [(ngModel)]="usuario.nome" placeholder="nome" type="text" >
<br>
<h1>Senha: </h1>
<input type="text" [(ngModel)]="usuario.senha" placeholder="senha">
<br><br>
<button mat-raised-button color="primary" class="button" (click)="cadastrar()" >Cadastrar</button>
```

Crie uma rota de acesso para o componente no arquivo app-routing.module.ts

Importe o component.

```
import {CadastrarComponent } from './cadastrar/cadastrar.component';
```

Adicione a lista de Rotas.

```
{ path: 'cadusuario', component: CadusuarioComponent },
```

Feito isso, execute no terminal “ng start” no servidor back-end e front-end.

Abra o endereço http://localhost:4200 no browser.

Crie uma rota de acesso para o componente no arquivo app-routing.module.ts

Importe o component.

```
import {CadastrarComponent } from './cadastrar/cadastrar.component';
```

Adicione a lista de Rotas.

```
{ path: 'cadusuario', component: CadusuarioComponent },
```

Abra o endereço http://localhost:4200/cadusuario

O browser abrirá o componente recém criado.

Faça um cadastro.

Código front-end completo
cadastrar.component.ts

```ts
import { Component, OnInit } from "@angular/core";
import { HttpService } from "../http.service";
import { ActivatedRoute } from "@angular/router";
import { Location } from "@angular/common";

@Component({
  selector: "app-cadastrar",
  templateUrl: "./cadastrar.component.html",
  styleUrls: ["./cadastrar.component.css"]
})
```

```ts
export class CadastrarComponent implements OnInit {
  usuario = {
    nome: "",
    senha: ""
  };

  constructor(
    public httpService: HttpService,
    private route: ActivatedRoute,
    private location: Location
  ) {
    this.httpService.location = location;
  }
  ngOnInit() {}
  cadastrar() {
    this.httpService.cadastrarUsuario(this.usuario);
  }
}
```

Código front-end completo cadastrar.component.html

```
<h1>Nome: </h1>
<input [(ngModel)]="usuario.nome" placeholder="nome" type="text" >
<br>
<h1>Senha: </h1>
<input type="text" [(ngModel)]="usuario.senha" placeholder="senha">
<br><br>
<button mat-raised-button color="primary" class="button" (click)="cadastrar()" >Cadastrar</button>
```

Código front-end completo listar.component.ts

```ts
import { Component, OnInit } from "@angular/core";
import { HttpService } from "../http.service";

export interface PeriodicElement {
  name: string;
  position: number;
  weight: number;
  symbol: string;
}

export interface Pessoa {
  name: string;
  senha: string;
}

@Component({
  selector: "app-listar",
  templateUrl: "./listar.component.html",
  styleUrls: ["./listar.component.css"]
})
```

```ts
@Component({
  selector: "app-listar",
  templateUrl: "./listar.component.html",
  styleUrls: ["./listar.component.css"]
})
export class ListarComponent implements OnInit {
  users = [];
  user;
  atualizar = false;
  constructor(private httpService: HttpService) {
    this.httpService = httpService;
    this.users = this.httpService.users;
  }

  ngOnInit() {
    this.httpService.getUsuarios();
  }
  deletar(usuario) {
    console.log("usuario.id" + usuario._id);
    this.httpService.deletarUsuario(usuario);
  }
  alterar(usuario) {
    this.user = usuario;
    this.atualizar = true;
  }
}
```

Código front-end completo listar.component.html

```
<h1>Listar Usuarios</h1>
<ul class="users">
  <li  *ngFor=" let user of users" >

    <span class="text">Usuario: {{user.nome}}    Senha: {{user.senha}}</span>

    <button  class="button1" mat-raised-button color="primary" (click)="deletar(user)">Deletar</button>
    <a routerLink="/atualizar/{{user._id}}">
      <button  class="button2" mat-raised-button color="primary" >Alterar</button>
    </a>

  </li>
</ul>
<a routerLink="/cadastrar">
  <button mat-raised-button color="primary">Cadastrar</button>
</a>
```

Código front-end completo atualizar.component.ts

```ts
import { Component, OnInit,Input } from '@angular/core';
import { HttpService } from '../http.service';
import { ActivatedRoute } from '@angular/router';
import { Location } from '@angular/common';

@Component({
  selector: 'app-atualizar',
  templateUrl: './atualizar.component.html',
  styleUrls: ['./atualizar.component.css']
})
```

```ts
export class AtualizarComponent implements OnInit {
  usuario;

  httpService: HttpService;
  constructor(
    httpService: HttpService,
    private route: ActivatedRoute,
    private location: Location
  ) {
    this.httpService = httpService;
    this.httpService.location = location;
  }
  ngOnInit() {
    const id = this.route.snapshot.paramMap.get("id");
    console.log("id= ", id);
    this.usuario = this.httpService.getUsuario(id);
  }
  atualizar() {
    this.httpService.atualizarUsuario(this.usuario);
  }
}
```

Código front-end completo atualizar.component.html

```
<h1>Nome: </h1>
<input [(ngModel)]="usuario.nome" placeholder="nome" type="text" >
<br>
<h1>Senha: </h1>
<input type="text" [(ngModel)]="usuario.senha" placeholder="senha">

<button mat-raised-button color="primary" class="button" (click)="atualizar()" >Atualizar</button>
```

Código front-end completo http.service.ts

```
import { HttpClient, HttpParams } from '@angular/common/http';
import { Injectable } from '@angular/core';

@Injectable()
export class HttpService {
    users = [];
    baseURL: string;
    flag = false;
    location;
    constructor(private http: HttpClient) {
        this.baseURL = 'http://localhost:3000/usuarios';
    }
```

Código front-end completo
http.service.ts

```
getUsuario(id) {
    let usuario;
    this.users.forEach(element => {
        if (element._id == id) usuario = element;
    });;
    return usuario;
}

getUsuarios() {

    this.http.get(this.baseURL)
        .subscribe((data: any) => {
            this.users.splice(0);
            data.forEach(element => {
                this.users.push(element);
                console.log(element);
            });
        })
}
```

```
cadastrarUsuario(usuario) {
    console.log("inspecao:" + this.baseURL + 'inserir');
    this.http.post(this.baseURL, usuario).subscribe((data: any) => {
        console.log("data resposta: " + data);
        this.getUsuarios();
        this.location.back();
    });
}

atualizarUsuario(usuario) {
    this.http.put(this.baseURL+'/'+usuario._id,
    {nome: usuario.nome, senha:usuario.senha})
    .subscribe((data: any) => {
        console.log("data resposta: " + data);
        this.getUsuarios();
        this.location.back();
    });

}
deletarUsuario(usuario) {
    console.log("usuario" + JSON.stringify(usuario));
    this.http.request('delete', this.baseURL+'/'+usuario._id, { body: usuario })
        .subscribe((data: any) => {
            this.getUsuarios();
        })
}
```

Obrigado!

Alguma pergunta?

Referências Bibliográficas

- Documentação do Angular
- https://angular.io
- Curso de Angular Loiane Groner
- https://www.youtube.com/channel/UCqQn92noBhY9VKQy4xCHPsg
- Projeto: https://github.com/gillallifam/DES_WEB