BIT PAGG LTDA.

ANGULAR

Araxá
2019

Sumário

1. Introdução 3
2. Aplicação angular 3
   2.1. Instalação Git 3
   2.2. Instalação node.js 4
   2.3. Instalação Angular CLI 4
   2.4. Gerando a aplicação 4
   2.5. Dentro da aplicação 5
   2.6. Criação de componentes 6
   2.7. Estrutura base do projeto 6
   2.8. Criando “Web.config” 7
   2.9. Simulando https no localhost 9
   2.10. Configuração da environments 9
   2.11. API REST 11
   2.12. Possíveis problemas 11
3. Boas práticas 12
   3.1. Projetos 12
   3.2. Não deve ser feito 13
   3.3. Segurança 14
   3.4. Parâmetros entre páginas 14
   3.5. Armazenamento de dados 16
   3.6. Consumo API pela Service 17
   3.7. Cors 19
4. Padrões Angular 19
   4.1. Padrões de desenvolvimento 19
   4.2. Erros 21
   4.3. Comandos mais utilizados “npm” 22

1. Introdução

Neste documento está centralizado as documentações de desenvolvimento inicial angular, desenvolvimento inicial proxy e as boas práticas adotas.

Todas as informações que estão contidas seguem como acordado com a equipe de suporte e marketing, segue também alterações no desenvolvimento e em padronização do desenvolvimento das aplicações.

2. Aplicação angular

2.1. Instalação Git

Primeiramente, é necessário instalar o **Git**, pois iremos usar o **Git CMD** como prompt para executar linhas de comando para a criação do projeto e instalação de ferramentas que iremos precisar para o projeto Angular.

Para a instalação do **Git** entre em (*https://git-scm.com/downloads*), faça o download do mesmo de acordo com seu sistema operacional, e execute-o para que ele seja instalado, não há configuração específica então e apenas seguir o fluxo de instalação.

Ao executar o instalador do Git irá começar o processo de instalação, a primeira página da instalação será para mostrar os termos da licença para utilização do **Git**, para prosseguir com a instalação aperte no botão **"*Next*"**.

Após a tela de licença irá se apresentar para a próxima seção de instalação para marcar os componentes necessários, que serão instalados. Em seguida clique em **"*Next*"** para continuar com a instalação.

Feito o procedimento acima irá abrir uma nova tela, essa nova tela será para selecionar o caminho onde será instalado o **Git**, por default aparecerá um caminho padrão caso queira trocá-lo vá em **"*Browse...*"** e escolha o caminho. Em seguida aperte **"*Next*"** para continuar o processo.

Em seguida abrirá a proxima tela, essa tela apresenta que será criado um atalho para que abra o **Git** dentro da pasta onde queira utilizá-lo, este atalho será exibido quando clicar com o botão direito do mouse. Clique em **"*Next*"** para prosseguir com a instalação.

Em seguida aparecerá a tela abaixo, onde pede para escolher como iremos utilizá-lo, o **Git** em geral e utilizado para efetuar comandos para Windows e para aplicação que criarmos, com isso marque a segunda opção como mostra abaixo e clique em **"*Next*"** para continuar com a instalação.

Em seguida aparecerá a tela abaixo que mostrara a opções de qual biblioteca de conexão HTTP, para essa opção escolha a padrão do **Git**, será a primeira opção. Em seguida aperte **"*Next*"** para prosseguir a instalação.

Em seguida selecione a primeira opção para a configuração de leitura de linhas de texto, como serão interpretadas pelo **Git**. Em seguida clique em **"*Next*"** para continuar a aplicação.

Em seguida selecione a primeira opção para que configuração do emulador do terminal **Git Bash**. Em seguida clique em **"*Next*"** para continuar a instalação.

Em seguida selecione as duas primeiras opções para que possam utilizar a credenciais necessárias para o funcionamento. Em seguida clique em **"*Install*"** para prosseguir com a instalação.

Ao término da instalação vá e procure o **Git,** abra ele como administrador, para que possa ter os privilégios necessários para utilizá-lo sem impedimentos.

2.2. Instalação node.js

Em seguida é necessário a instalação do **node.js** (https://node.js/en/download). Efetue o download de acordo com o seu sistema operacional e execute-o para a instalação apenas siga o fluxo de instalação sem configurações especificas.

Após executá-lo iniciará o processo de execução, clique em **"*Next*"** para continuar o procedimento.

Em seguida clique em aceitar os termos de uso e em **"*Next*"** novamente.

Em seguida escolha o diretório onde será instalado o node, caso queira mudar o caminho clique em **"CHANGE"** e escolha o caminho de sua preferência, em seguida clique em **"*Next*"** para continuar o processo de instalação.

A próxima tela mostra a as bibliotecas que irão ser instaladas, nesta tela apenas clique em **"*Next*"** para continuar a instalação.

Em seguida clique em **"*Install*"** para finalizar a instalação.

Após finalizar a instalação clique em **"*Finish*"** para fechar a execução de instalação.

Após feita a instalação abra o **Git** como administrador e execute o seguinte comando **"node -v"**, para ver a versão do node instalada.

2.3. Instalação Angular CLI

Após feito esse procedimento de instalação do **node.js**, vamos instalar o **Angular CLI**, que é usado para a criação de aplicações, e componentes a serem utilizados dentro da aplicação.

Para a instalação do **Angular CLI**, é necessário abrir o **Git CMD** como administrador.

Em seguida escreva o comando **"npm install -g @angular/cli"**.

Após o término do processo de instalação digitamos o seguinte comando para verificarmos se a instalação foi bem-sucedida **"ng -v"**, irá abrir um sumário descrevendo as configurações da instalação do **Angular CLI**.

2.4. Gerando a aplicação

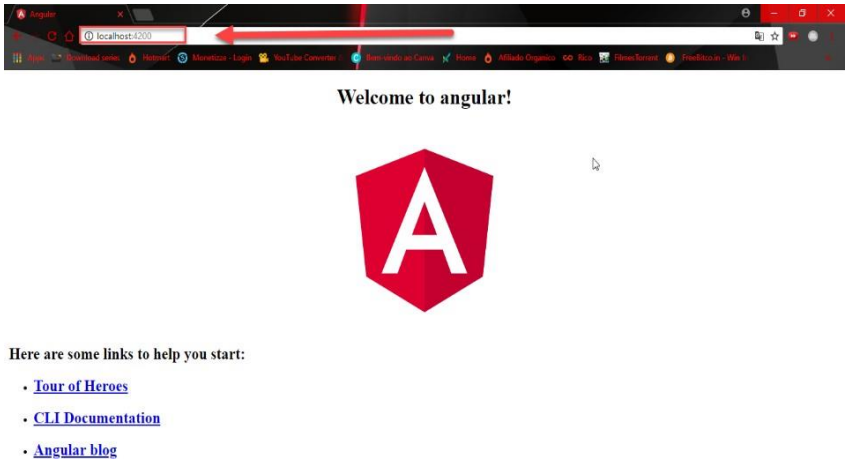
Para gerar uma aplicação é necessária a abertura do **Git CMD** como administrador, pois iremos criar a aplicação por linhas de comando.

Para criar a aplicação é necessário que naveguemos pelo **Git CMD** até a pasta onde será criada a aplicação usando o comando **"cd.."** para voltar uma pasta e **"cd'NomeDaPasta'\"** para entrar dentro da pasta.

Quando chegar na pasta que criada para armazenar a aplicação você irá executar o seguinte comando para criar a aplicação **"ng new 'NomeDaAplicação' --prefix=jad"**, ex: "ng new AplicaçãoAngular --prefix=jad", feito este procedimento levará alguns minutos para gerar a aplicação e instalar as dependências necessárias.

Após o término da criação do projeto navegue até a pasta do projeto criado, em seguida para verificar sua aplicação, e necessário um servidor local, para dar um starter no servidor de desenvolvimento, execute o seguinte comando **"ng serve"**, ele irá subir sua aplicação em uma porta padrão **"localhost:4200"**, e, ao efetuar a inicialização do servidor, vá no seu browser e digite **"http://localhost:4200"** para ver sua aplicação.

Inicialmente na página do projeto estará um menu sobre o angular, com alguns links para navegação. Como a imagem abaixo.



Para a edição seu código é necessário uma **IDE**, usaremos o **Visual Studio Code**, para que possam ser feitas as alteração e desenvolvimento da aplicação.

2.5. Dentro da aplicação

Vamos passar por alguns dos componentes mais importantes que iremos utilizar para o desenvolvimento da aplicação.

Entre no **Visual Studio Code** e abra a pasta da aplicação criada, após abri-la vemos a estrutura criada, quando criarmos a aplicação no **Git**, na estrutura principal vemos os seguintes arquivos:

- **Package.json**, que é o arquivo de configuração das dependências do projeto;
- Pasta **src**, que contém os seguintes arquivos:
  - **Style.css**, este arquivo irá conter todos os styles globais da sua aplicação;
  - **Polyfills.ts**, ele irá conter todos scripts que irão dar suporte à novas funcionalidades, caso browsers não executem as filters mais recentes do Java script;

- **Index.html**, que está dentro da pasta environments, onde está o arquivo principal html;
- **Main.ts**, que é usado para startar o bootstrap da aplicação, usando os arquivos na pasta 'app' que está dentro da pasta 'src', onde estão as telas e componentes da aplicação;
- Pasta **App**, onde estão os arquivos .ts, .html e .css:
- **App.module.ts**, que é um arquivo de módulo onde ficam as declarações de todos os componentes;
- **App.component.ts**, que são arquivos para construção de classes e diretrizes do componente;
- **App.component.css**, arquivo para construção de css do componente;
- **App.component.html**, arquivo para a estruturação do componente em html;
- **App.component.spec.ts**, arquivo de testes do componente.

2.6. Criação de componentes

Para a criação de um componente dentro da aplicação é necessária a abertura de uma nova aba do **Git CMD** como administrador.

Navegue até a pasta app para criarmos o novo componente. Chegando dentro da pasta execute o seguinte comando para a criação do componente: **"ng g c 'NomeDoComponente' --spec=false"**.

Exemplo: **"ng g c TesteComponente -- spec=false"**, o **"--spec=false"** é um comando para que não se criem os componentes de teste, caso queira criar o seguinte componente de teste coloque **"true"**.

Ao finalizar a criação do componente, vá em sua IDE e entre na pasta **'app'** e confira se foi criado corretamente. Deverá ter sido criada uma pasta com o nome do seu componente e dentro dessa pasta terão sido criados três arquivos: um **'componente.css'** onde ficará o style do seu componente, o **'componente.html'** onde ficará o template do componente e o **'componente.ts'** que é a classe do seu componente.

O **Angular CLI,** ao criar uma componente via linha de comando, já cria a chamada de nosso componente dentro do **'app.modulo.ts'**, onde fica a chamada de todos os componentes.

2.7. Estrutura base do projeto

Para facilitar a manutenção e entendimento do projeto, iremos trabalhar com uma estrutura pré-definida de pastas, aonde cada base será responsável pela divisão de responsabilidades e conexões no projeto.

- ***Components***

Criação de elementos que compõem o layout do site e que podem ser reaproveitados em várias partes do projeto.

Exemplo de comando: ng g c componentes/file-upload

- ***Modules***

Responsável para receber os demais componentes e forms do projeto, assim montando a estrutura das páginas do projeto, separado pelo nome dos módulos

Exemplo de comando: ng g c modules/registerAccountContainer

- ***Views***

Devemos gerar as views para serem criadas as rotas do projeto, sendo assim o único elemento HTML que deve conter é o Container correspondente ao mesmo.

Exemplo de comando: ng g c views/registerAccountViews

- ***templates***

Base de estrutura do layout aonde será exibido em todas as páginas.

Exemplo de código:

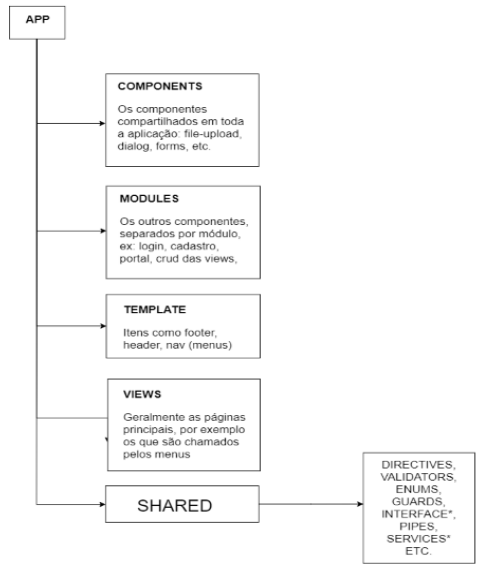
```
<div class="dashboardLayout">
  <div class="container-fluid">
    <div class="row">
      <div class="col-md-1">
        <app-sidebar></app-sidebar>
      </div>
      <div class="col-md-11">
        <div class="dashboardLayout_content">
          <router-outlet></router-outlet>
        </div>
      </div>
    </div>
  </div>
</div>
```

Exemplo de comando: ng g c layouts/dashboard

- ***shared***

Módulo separado para organizar os itens de configuração da aplicação como: directives, pipes, validators, enums, guards, interfaces e services.

ARQUITETURA DE SISTEMAS



Interfaces

ressalva aqui, costumo usar bastante no lugar das classes, pois não precisam nem podem ser instanciadas, o resultado da transpilação, pegar seu código e converter ele para outro em tempo de compilação, não gera código JavaScript isso é importante porque diminui o tamanho do build e melhora o tempo de processamento para tarefas como concat e minify), ou seja só uso classe, mesmo quando preciso fazer um parse dos dados vindos do servidor..

Serviços

Eu costumo colocar separado os serviços, mas se os serviços estiverem atribuídos à telas, como um de/para aí coloco dentro dos componentes mesmo

| - app

| - Components

| - [+] file-upload

| - [+] confirm-dialog

| - [+] alert-upload

| - [+] file-upload

| - [+] breadcrumb

| - views

| - [+] home

| - [+] product-crud

| - [+] client-crud

| - templates

| - [+] header

| - [+] nav

| - [+] footer

| - modules

| - register

| - [+] contact

| - [+] data

| - portal

| - product

| - [+] product-edit

| - [+] product-create

| - [+] product-delete

| - client

| - [+] client-edit

| - [+] client-create

| - [+] client-delete

| - [+] usecases

| - shared

| - [+] commons

| - [+] validators

| - [+] directives

| - [+] enums

| - [+] guards

| - [+] interfaces

| - [+] pipes

| - [+] services

| - shared.module.ts

| - app.component.html

| - app.module.ts

| - app-routing.module.ts

| - app.component.ts

Nos módulos podemos utilizar o modelo lazy load, neste modelo o aplicativo não precisa carregar tudo de uma só vez. Perfeito para aplicações grande, todo o conteúdo do módulo está diretamente relacionado com a rota.

|-- pedido // Module

|-- novo

|-- novo-pedido.component .|html|scss|spec|.ts

|-- editar

|-- editar-pedido.component .|html|scss|spec|.ts

|-- detalhes

|-- detalhes-pedido.component .|html|scss|spec|.ts

|-- pedido.service.ts

|-- pedido.module.ts

|-- pedido.routes.ts

Trabalhar com módulos há ganhos significativos na inicialização da aplicação. É inimaginável uma aplicação Angular de médio porte sem módulos lazy load. E essa estrutura é coesa, mantém os grupos de interesse dentro de um mesmo local. (Bruno Brito)

2.8. Criando "Web.config"

Será necessário a criação de um arquivo web.config no Front end pois algumas vulnerabilidades da aplicação serão corrigidas através deste arquivo.

Para a configuração da Web.config no ambiente de desenvolvimento será necessário a criação do arquivo **"Web.config"** na raiz do projeto para que seja criado os parâmetros dentro deste arquivo.

Deverá seguir o seguinte modelo:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <httpProtocol>
      <customHeaders>
        <remove name="X-Powered-By"/>
        <add name="X-Frame-Options" value="SAMEORIGIN"/>
        <add name="Cache-Control" value="no-cache, no-store, must-revalidate"/>
        <add  name="X-XSS-Protection"  value="1;mode=block"/>
        <add  name="X-Content-Type-Options" value="nosniff"/>
        <add name="Pragma" value="no-cache"/>
      </customHeaders>
    </httpProtocol>
    <rewrite>
      <rules>
        <clear />
        <rule name="HTTP to HTTPS" stopProcessing="true">
          <match url="(.*)"/>
          <conditions>
            <add input="{ALL_HTTP}" matchType="Pattern" pattern="HTTP_X_FORWARDED_PROTO:https" ignoreCase="true"
negate="true"/>
          </conditions>
          <action type="Redirect" url="https://{HTTP_HOST}/{R:1}" />
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
  <location path="Mapfre">
  </location>
  <system.web>
    <sessionState timeout="20"/>
  </system.web>
</configuration>
```

Neste modelo contém os parâmetros para correção das vulnerabilidades observadas na aplicação, e existem algumas regras criadas para configuração da URL da aplicação essas regras deverão ser imutáveis sendo assim deverão ser iguais ao do modelo acima, foram usados os seguintes parâmetros para correção das vulnerabilidades detectadas no cabeçalho de resposta do servidor:

- **<add name="X-Frame-Options" value="SAMEORIGIN"/>**

- **<add name="Cache-Control" value="no-cache, no-store, must-revalidate"/>**

- **<add name="X-XSS-Protection" value="1; mode=block"/>**

- **<add name="X-Content-Type-Options" value="nosniff"/>**

- **<add name="Pragma" value="no-cache"/>**

Em outros projetos que não são referentes a Mapfre favor retirar a seguinte parâmetro, pois não será necessário a utilização:

- **<location path="Mapfre"> </location>**

Para adicionar o arquivo automaticamente no build do angular, devemos criar o arquivo dentro da pasta SRC.

Abra o arquivo angular.json:

```
"assets": [
            "src/favicon.ico",
            "src/assets",
            "src/web.config"
          ]
```

2.9. Simulando https no localhost

Para iniciar o projeto simulando um https no servidor angular, inicie o projeto com o comando abaixo.

Comando: ng serve –ssl

2.10. Configuração da environments

Para configuração da URL do Endpoint para consumo dos serviços, devemos utilizar a configuração do arquivo environments.ts.

Nesse arquivo devemos adicionar a URL local de desenvolvimento:

```
export const environment = {
    production: false,
    API_URL: 'http://localhost:3000/node/projeto',};
```

Como temos os ambientes de Homologação e Produção, precisamos criar mais dois arquivos:

- *environments.prod.ts*
- *environments.hml.ts*

Em cada arquivo deve conter a URL do Micro Serviço respectivo para cada ambiente.

Para efetivar a troca de URL para cada vez que geramos um novo build, devemos configurar no arquivo angular.json da seguinte maneira:

```
"configurations": {
        "production": {
         "fileReplacements": [
          {
            "replace": "src/environments/environment.ts",
            "with": "src/environments/environment.prod.ts"
          }
         ],
         "optimization": true,
         "outputHashing": "all",
         "sourceMap": false,
         "extractCss": true,
         "namedChunks": false,
         "aot": true,
         "extractLicenses": true,
         "vendorChunk": false,
         "buildOptimizer": true,
         "budgets": [
          {
            "type": "initial",
            "maximumWarning": "8mb",
            "maximumError": "8mb"
          }
         ]
        },
        "homolog": {
         "fileReplacements": [
          {
            "replace": "src/environments/environment.ts",
            "with": "src/environments/environment.hml.ts"
          }
```

```
      ],
      "optimization": true,
      "outputHashing": "all",
      "sourceMap": false,
      "extractCss": true,
      "namedChunks": false,
      "aot": true,
      "extractLicenses": true,
      "vendorChunk": false,
      "buildOptimizer": true,
      "budgets": [
       {
        "type": "initial",
        "maximumWarning": "8mb",
        "maximumError": "8mb"
       }
      ]
     }
    }
```

Comando para gerar o build para HML: ng build –output-hashing=all –configuration=homolog

Comando para gerar o build para PRD: ng build –output-hashing=all –configuration=production

2.11. API REST

Para a criação de um arquivo.json (db.json) para consultas backend é necessária abertura do o Git CMD como administrador.

E navegação até a pasta raiz da aplicação, digite o seguinte comando **"npm install -g json-server"**, após o comando levará alguns minutos até que o serviço seja criado.

Após o término do comando de instalação, execute o comando **"json-server db.json"**, este comando retornará a URL de endereço de seu arquivo json, inicialmente com um arquivo default em **"db.json"**.

Ao utilizar as URL´s determinadas, abra em seu browser e veja o que há dentro dos arquivos.

Para abrir a arquivo json vá em sua IDE e procure na raiz de seu projeto o arquivo **"db.json"**, neste arquivo você colocará o conteúdo de seu json. Ele inicia com um arquivo default.

2.12. Possíveis problemas

Requisições:

As chamadas de API não estão sendo autenticadas em nenhum momento, pois não é passado o **"x-access-token"** nos cabeçalhos, bem como o próprio token gerado pelo método ***".../autenticador/token"***.

Chamadas de API não estão de acordo com a documentação, desenvolvida e enviada em 26/09, estão sendo feitas diretamente, sendo que deveriam ser utilizadas uma **"service"** para realizar as solicitações a proxy.

Reload Página:

Para a resolução do problema de **"Not Found"** em relação ao carregamento da página, vá em **"app.module.ts"** realize o seguinte import:

"import {HashLocationStrategy, LocationStrategy} from '@angular/common'"

Em seguida, dentro de **"@NgModule"**, vá em **"providers"**, inclua os paramêtros **"HashLocationStrategy, LocationStrategy"** para utilização da biblioteca no projeto.

3. Boas práticas

3.1. Projetos

Para a utilização de angular em alguma aplicação existe algumas boas práticas que irão auxiliar no desenvolvimento do projeto, tendo uma melhor organização no decorrer do projeto.

Para iniciar um projeto angular é ideal utilizar o **"prompt"**, efetue instalação do **"angular cli"** globalmente com o comando **"npm install -g @angular/cli"** em seguida para gerar o projeto execute o seguinte comando **"ng new 'nome-do-projeto'"**, e será criado as sequências de pastas da raiz do projeto, para a criação de componentes no decorrer do projeto e aconselhável que seja feito através do comando **"ng g c 'nome-componente'"**, pois irá ser criado um componente com todas as dependências necessárias sem se preocupar se está faltando algo, e também na criação de uma service com o comando **"ng g service 'nome-service'"**.

Em relação aos componentes crie um padrão para nomenclatura para os mesmos, e claro que isto se aplica para qualquer tipo de código, seja para evitarmos números mágicos arbitrários ou para evitar a tentação de abreviar variáveis. Em um mundo perfeito, seus componentes terão nomes que são bem legíveis e fluem como um texto quando colocados no seu código. Quando for registrar o nome dos variáveis e/ou funções, utilize o estilo **camelCase** para ser algo bem legível: "**minhaVariavel**", "**funcionalidadeTipo**". Já para Classes e Interfaces utilize o estilo **PascalCase "MinhaClasse"**, é simples, fácil e faz sentido do ponto de vista lógico. Você pode escolher outra convenção, mas o que importa é ser consistente e ter algum tipo de método no meio da loucura.

Para melhor organização de seus componentes e aconselhável fazer o seguinte, apenas um componente por arquivo, tentando manter este arquivo com menos de 400 linhas de código. Isto faz o seu código substancialmente mais fácil de ler, navegar e manter. Isto previne conflitos ao utilizar um SCM como o Git e problemas com declaração de variáveis repetidas e coisas semelhantes.

Quando houver a necessidade de criar alguma classe, interface ou serviço que será utilizado em toda a aplicação e em qualquer momento, e aconselhável que efetue a criação de uma pasta referente ao serviço ou a classe que irá utilizar, pois ficar mais fácil a localização dos mesmos e melhora a organização das pastas na raiz do projeto.

Sempre que possível utilize interface no lugar de classes, pois não precisam e nem podem ser instanciadas, o resultado da transpilação, "pegar seu código e converter ele para outro em tempo de compilação", não gera código JavaScript isso é importante porque diminui o tamanho do build e melhora o tempo de processamento para tarefas como concat e minify), ou

seja só usar classe, mesmo quando preciso fazer um parse dos dados vindos do servidor.

Crie pasta com nome da feature que ela representa, a estrutura fica mais legível e não existem nomes redundantes e repetitivos. O desenvolvedor consegue localizar rapidamente o código e identificar o que cada arquivo representa.

É sempre uma boa prática extrair sua lógica de negócios principal para os serviços. Dessa forma, fica muito mais fácil de manter, pois pode ser trocado e substituído por uma nova implementação em apenas alguns segundos.
Isso vale para testes. Muitas vezes você precisa de serviços que buscam dados externos para falsificar os resultados em um ambiente de teste. Se você buscar seus dados nos serviços, isso é fácil. Se não, terá que mudar todas as linhas que precisam ser alteradas para isso.

Falando de serviços, quando for criar um utilizaremos promisses com o async/await, pois com subscribe "observable", pode ocorrer vazamento de memória e problemas de performance da aplicação.

```
getProduct() {
  this.productService.read().subscribe(products => {
    this.products = products;
    console.log(products);
  })
}
```

Com async / await

```
async getProduct() {
  this.products = await this.productService.read();
  console.log(this.products);
}
```

O controle de erro "try/catch" pode ser feito no serviço, ou no componente quando utilizar várias requisições.

Mantenha o tamanho do seu aplicativo pequeno importando apenas o que você precisa, cada declaração de importação que você usa aumenta o tamanho do seu pacote, tendo algumas bibliotecas muito grandes e ao usar uma declaração de importação incorreta, você pode acabar com toda a biblioteca de seu aplicativo.

O Angular não usa diretivas tanto quanto o Angular.js, mas ainda tem algumas como: *ngIf*, *ngFor. Então sempre que necessário usar estas diretivas pois com certeza irá aumentar o desempenho de sua aplicação.*

Utilizar interpolação ao concatenar strings:
Trocar de "environment.API_URL + '/authenticator/ticket'" para
`${environment.API_URL}/authenticator/ticket`

3.2. Não deve ser feito

Iremos falar um pouco do que não e aconselhável fazer em aplicações angular, para evitar erros e melhorar desempenho e manutenibilidade de sua aplicação angular.

Os componentes são os blocos de construção essenciais em um ecossistema angular, a ponte que conecta a lógica de nosso aplicativo com a visão. Mas às vezes os desenvolvedores ignoram fortemente os benefícios que um componente oferece.

Pode acontecer de criarem um código dentro de um componente que se repete em outros componentes ou até mesmo no mesmo componente, então o ideal e transformar essa parte do código que se repete em componente individual, para melhor acesso a aplicação.

Às vezes, você pode tender a pensar nos dados trazidos de um servidor / API como *qualquer*. Isso não é realmente o caso, você deve definir os tipos de dados que você recebe do seu back-end, para otimizar o uso dos dados em seus componentes por isso que o angular opta por ser usado principalmente no TypeScript.

Eu sugiro não fazer isso em um serviço, pois os serviços são para chamadas de API, compartilhando dados entre componentes e outros utilitários. As manipulações de dados devem

pertencer aos componentes e modelos separados.

*E ao criar um serviço crie o mesmo na pasta **"app"** da aplicação, pois todos os outros subcomponentes criados dentro da pasta **"app"** poderão utilizar a mesma informação que o serviço irá trazer, e com alerto que não e aconselhável criar mais de uma service na aplicação, e criar dentro das pastas dos subcomponentes com as mesmas funcionalidades do serviço principal, pois pode haver algum tipo de conflito, e a estrutura de pastas ficará desorganizada para futuras manutenções e implementações.*

Cuidado ao declarar o seu componente no **"AppModule"**, pois ele não deve ser declarado mais de uma vez na module, por isso aconselho a criar o componente através de linhas de comando pelo prompt, pois ao criar automaticamente o comando já declara a instância do componente no **"AppModule"**.

3.3. Segurança

Iremos falara sobre algumas boas práticas em relação a desenvolvimento de sua aplicação.

Uma boa prática para a segurança, recomenda-se atualizar as bibliotecas Angular em intervalos regulares. Não o fazer pode permitir que invasores ataquem o aplicativo usando vulnerabilidades de segurança conhecidas presentes em versões mais antigas.

Em relação a comunicação do front-end e back-end pode-se usar o jwt, que e usado para codificar e decodificar token, usando este token para efetuar validações de usuário e transferência de informações via chamadas Rest.

Previna o script entre sites (XSS) que permite que invasores injetem códigos maliciosos em páginas da Web. Esse código pode, por exemplo, roubar dados do usuário (em particular, dados de login) ou executar ações para representar o usuário. Este é um dos ataques mais comuns na web.

Para bloquear ataques XSS, você deve impedir que códigos maliciosos entrem no DOM (Document Object Model). Por exemplo, se os invasores puderem induzi-lo a inserir uma tag <script> no DOM, eles poderão executar um código arbitrário em seu website. O ataque não está limitado a tags <script> - muitos elementos e propriedades no DOM permitem a execução de códigos, por exemplo, <img onerror = "..."> e <a href="javascript:..."> . Se dados controlados pelo invasor entrarem no DOM, espere vulnerabilidades de segurança.

Previnir as APIs do navegador incorporado, pois não protegem você automaticamente contra vulnerabilidades de segurança. Por exemplo, documento, o nó disponível por meio do ElementRef e muitas APIs de terceiros contêm métodos não seguros. Da mesma forma, se você interagir com outras bibliotecas que manipulam o DOM, provavelmente não terá a mesma sanitização automática das interpolações angulares. Evite interagir diretamente com o DOM e, em vez disso, use modelos angulares sempre que possível.

Para casos em que isso é inevitável, use as funções de higienização angular incorporadas. Sanitize valores não confiáveis com o método DomSanitizer.sanitize e o SecurityContext apropriado. Essa função também aceita valores que foram marcados como confiáveis usando as funções bypassSecurityTrust ... e não os higienizará, conforme descrito abaixo.

Ter cuidado ao incluir código executável, exibir um <iframe> de algum URL ou construir URLs potencialmente perigosos. Para evitar a sanitização automática em qualquer uma dessas situações, você pode informar ao angular que você inspecionou um valor, verificou como ele foi gerado e garantiu que ele estivesse sempre seguro. Mas tenha cuidado, se você confia em um valor que pode ser mal-intencionado, está introduzindo uma vulnerabilidade de segurança em seu aplicativo.

3.4. Parâmetros entre páginas

Para efetuar a transição de parâmetros entre as páginas da aplicação existem maneiras diferentes de fazer isso, mais precisamente iremos abordar 3 dessas maneiras.

A primeira que iremos desenvolver será através de uma função que criaremos, nesta função usaremos a **“queryParam”** para passarmos os valores dos parâmetros para a outra da página de nossa aplicação, o único empecilho que está formato de solução e que ele além de passar os valores nas variáveis ele também exibe na URL da rota. Então não e aconselhado usar este método por não ter segurança das informações.

A princípio devemos importar a biblioteca de rotas do angular e navegação, **“@angular/router”**.

Em seguida devemos declarar uma variável do tipo **"Router"** no construtor, que irá disponibilizar as opções de métodos da biblioteca.

Então criamos nosso método que irá pegar as variáveis colocadas em uma variável do tipo **"NavigationExtras"**, dentro do parâmetro **"queryParams"** onde o conteúdo das variáveis estará armazenado.

Em seguida informamos para qual rota vai ser passado os parâmetros que estão em **"NavigationExtras"**, com isso ao chamar esta função em um evento de tela qualquer ela mudará para a tela correspondente com a rota que foi colocada, e na URL aparecerá a rota concatenado com as informações dos parâmetros.

Na segunda página para onde você direcionou os parâmetros você de importar a biblioteca **"@angular/router"** como na primeira página.

Em seguida declare as variáveis que irão receber os parâmetros vindos da página anterior dentro da classe do componente.

Em seguida devemos declarar uma variável do tipo **"Router"** no construtor, que irá disponibilizar as opções de métodos da biblioteca.

Então para receber os parâmetros devemos chamara a **"queryParams"** de **"route"** e sobrescrever os parâmetros nas variáveis que determinamos para receber as informações.

A segunda opção para passar os parâmetros para outra página e usando o cookie do navegador, neste método armazenamos os parâmetros no cookie do navegador e depois damos um **"get"** para chamarmos os mesmos para utilizarmos.

Antes de começar a implementar transferência vai prompt de comando e execute o seguinte comando **"npm install ngx-cookie-service --save"** para utilizar a biblioteca de armazenamento no cookie, em seguida vá em **"app. modules"** e faça o seguinte, importe o **"CookService"** da biblioteca **"ngx-cookie-service"**.

Em seguida coloque ele na lista de providers, para que possamos usá-la no resto da aplicação, pois **"CookService"** irá fornecer um serviço para o resto da aplicação.

Na primeira página onde será recolhido o parâmetro primeiramente devemos importar a biblioteca para utilizarmos os serviços do cookie do navegador.

Em seguida declarar uma variável do tipo **"CookieService"** para utilizarmos suas dependências.

Em seguida vá em **"ngOnInit()"** e damos um **"set"** em **"cookieService"** para armazenarmos na variável **"Test"** a string **"Hello world"**, onde será gravada no cookie do navegador.

Na segunda página onde irá utilizar a variável que armazenou no cookie, primeiramente devemos importar a biblioteca para utilizarmos os serviços do cookie do navegador, como na primeira página.

Em seguida declarar no construtor uma variável do tipo **"CookieService"** para utilizarmos suas dependências.

Em seguida declare a variável que irá receber o parâmetro que foi gravado, dentro da classe do componente.

Em seguida vá em **"ngOnInit()"** e dê um **"get"** em **"cookieService"** com nome do parâmetro onde foi gravado no cookie, e armazene o retorno na variável que declarou.

O terceiro método para transferir parâmetros entre páginas e usar uma classe de variáveis para efetuar essa transição, ou seja, criando uma classe para ser utilizada como ponte, armazenando os valores necessários e depois buscando onde e quando necessário.

Primeiro e necessário criar uma pasta dentro de "app" e dentro desta pasta criar um arquivo **"nomearquivo"**.ts, e dentro deste arquivo será criada uma classe, dentro desta crie duas funções uma **"get"** e a outra **"set"**. A função **"set"** irá armazenar os valores dentro da **"variável"** e a **"get"** terá a função de fornecer o valor da **"variável"** quando for chamado.

Em seguida na primeira página onde for recolher as variáveis, deverá ser importado a classe **"nameModulo"**, está na **"app.modulo"**.

Em seguida deve-se declarar o nome da classe na lista de **"providers"** do componente, pois iremos utilizar um serviço que vem de fora do componente, depois declarar no construtor do componente uma variável do tipo **"nameModulo"** para herdar todas as funções que estão na classe.

Em seguida na função **"ngOnInit()"** do componente da página onde irá recolher os dados, chame a variável que declarou no construtor do componente e atribuir a função **".set()"** que e herdada da classe e colocar a variável que deseja armazenar para chamar em outra página.

Sendo assim no componente da segunda página onde pretende utilizar o valor da variável que foi guardada dentro da classe, e necessário primeiro importar a mesma classe que usamos na página anterior.

Em seguida repetir os mesmos passos que fizemos na página anterior, declarar o nome da classe na lista de **"providers"** do componente, pois iremos utilizar um serviço que vem de fora do componente, depois declarar no construtor do componente uma variável do tipo para herdar todas as funções que estão na classe.

Em seguida na função **"ngOnInit()"** do componente da página onde irá recolher os dados, chame a variável que declarou no construtor do componente e atribuir a função **".get()"** que é herdada da classe e armazene o retorno desta função na variável que deseja utilizar para manipular dentro da página.

3.5. Armazenamento de dados

Como vimos no tópico anterior podem ser usadas diferentes formas para armazenamento de variáveis dentro da aplicação, mas algumas delas devem ser tratadas ao término do uso do usuário e/ou ao fechamento da aplicação, para que não haja informações que possam prejudicar e expor o usuário.

A session e os cookies são muito utilizados nas aplicação para segurar o a seção do usuário dentro da aplicação, mas por padrão foi definido que ao fechamento da aba da aplicação ou em qualquer outro caso que haja retenção de dados do usuário ao termino da utilização da aplicação deve ser limpa e apagado qualquer tipo de dado retido na seção ou em cookies.

Para o armazenamento de dados na session usa-se o localStorage e SessionStorage, para a limpeza dos mesmos e necessário a utilização das seguintes funções, "**localStorage.removeItem('NomeDaVariavel')**","**localStorage.clear()**", "**sessionStorage.removeItem('NomeDaVariavel')**", "**sessionStorage.clear()**", com estas funções e possível a limpeza dos dados armazenados.

Para o armazenamento de dados utilizando os cookies no browser também e necessário a limpeza dos mesmos, para efetuar esta limpeza e utilizado as seguintes Bfunções, "**Cookie.delete('NomeDaVariavel');**","**Cookie.deleteAll();**", com estas funções e possível efetuar a limpeza dos cookies no navegador.

O armazenamento em cookie pode ser acessado pelo usuário a qualquer momento, então não e aconselhável o uso dos cookies para trafegar dados sensíveis na aplicação, porém a maneiras de efetuar a gravação nos cookies limitando o acesso aos dados gravados.

A utilização do **"Secure"**, parametro que garante o cookie só pode ser transmitido com segurança por HTTPS e não será enviado por conexões http não criptografadas.

E também pode ser usado o **"HttpOnly"** que torna o cookie inacessível por meio do **"document.cookie"** API, portanto, eles são editáveis apenas pelo servidor.

Objetivos das funções citadas:

Função	Funcionalidade
localStorage.removeItem('NomeDaVariavel')	Esta função tem a funcionalidade de apagar o conteúdo armazenado em localStorage com nome específico citado.
localStorage.clear()	Esta função tem a funcionalidade de apagar todos os conteúdos armazenado em localStorage, referente a todas as variáveis criadas.
sessionStorage.removeItem('NomeDaVariavel')	Esta função tem a funcionalidade de apagar o conteúdo armazenado em sessionStorage com nome específico citado.
sessionStorage.clear()	Esta função tem a funcionalidade de apagar todos os conteúdos armazenado em sessionStorage, referente a todas as variáveis criadas.
Cookie.delete('NomeDaVariavel')	Esta função tem a funcionalidade de deletar o conteúdo armazenado em Cookies com nome específico citado.
Cookie.deleteAll()	Esta função tem a funcionalidade de apagar todos os conteúdos armazenado em Cookies, referente a todas as variáveis criadas.

3.6. Consumo API pela Service

Para o consumo de uma API ou proxy, e necessário efetuar a criação de uma service, para a criação da service primeiro crie uma pasta com o nome **"service"** e navegue até a pasta usando o prompt, chegando dentro da pasta execute o seguinte comando **"ng g service 'nome_service' --spec"**, então será criado a service.

Após criado a service por default ela virá com alguns elementos. Já crida a classe onde será feita as chamadas externas.

Para efetuar as chamadas Http/Rest, GET, POST, PUT e DELETE, você deverá importar a biblioteca http do angular **"@angular/common/http"**, para efetuar as chamadas via url.

Em seguida declare uma variável no construtor da classe **"AppService"** que irá receber **"HttpClient"**, que herda as dependências da biblioteca http importada.

Agora vamos criar as funções referentes as chamadas rest, para exemplificar a função GET, usaremos uma função chamada **"ping()"**, nesta função usamos a variável **"http"** que declaramos no construtor, para utilizarmos a função **".get()"** que vem da biblioteca **"http"** do angular, e dentro do dos parênteses da função get e declarado a url, da API ou proxy que será feita a chamada. Então a função **"ping()"** retornará a resposta da chamada.

Para exemplificar a função POST executa o processo que está no corpo como um subordinado da URL, usaremos uma função chamada **"Login(inBody, inHeader)"**, nesta função usamos a variável **"http"** que declaramos no construtor, para utilizarmos a função **".post()"** que vem da biblioteca **"http"** do angular, e os parâmetros passados na função serão usados para transferir os dados no corpo da chamada e no cabeçalho, então as informações pertinentes a esses dois lugares deverão estar dentro dos parênteses quando chamada à função **"Login()"**, e dentro do dos parênteses da função **".post()"** e declarado a url da API ou proxy que será feita a chamada, e após a url e onde fica os parâmetros de inbody e inHeader que receberão as informações na chamada da função. Então a função **"Login()"** retornará a resposta da chamada. Para exemplificar a função PUT cria ou atualiza uma informação identificada por uma URL, usaremos uma função chamada **"CancelarVenda(inHeader, url)"**, nesta função usamos a variável **"http"** que declaramos no construtor, para utilizarmos a função **".put()"** que vem da biblioteca **"http"** do angular, e os parâmetros passados na função serão usados para transferir os dados na url da chamada e no cabeçalho, então as informações pertinentes a esses dois lugares deverão estar dentro dos parênteses quando chamada à função **"CancelarVenda()"**, e dentro do dos parênteses da função **".put()"** e declarado a url da API ou proxy que será feita a chamada, e após a url e onde fica os parâmetros de inbody e inHeader que receberão as informações na chamada da função. Então a função **"CancelarVenda()"** retornará a resposta da chamada.

Para exemplificar a função DELETE que deleta informação identificada por uma URL, usaremos uma função chamada **"DeleteCpf(inHeader, url)"**, nesta função usamos a variável **"http"** que declaramos no construtor, para utilizarmos a função **".delete()"** que vem da biblioteca **"http"** do angular, e os parâmetros passados na função serão usados para transferir os dados na url da chamada e no cabeçalho, então as informações pertinentes a esses dois lugares deverão estar dentro dos parênteses quando chamada à função **"DeleteCpf()"**, e dentro do dos parênteses da função **".delete()"** e declarado a url da API ou proxy que será feita a chamada, e após a url e onde fica os parâmetro inHeader que recebera as informações na chamada da função. Então a função **"DeleteCpf()"** retornará a resposta da chamada.

Para receber o retorno das funções acima dentro dos componentes para podermos utilizar a informações que vem da API, fazemos da seguinte maneira, primeiro importamos a classe **"AppService"** de **"app.service"** para podermos chamar as funções que utilizamos para efetuar as chamadas rest.

Em seguida declaramos a classe **"AppService"** na lista de providers pois é um serviço externo ao componente, e em seguida declaramos uma variável do tipo **"AppService"** para herdar todas as funções que estão na classe.

Em seguida para filtrarmos o retorno da função seja ela GET, PUT, POST ou DELETE.

Onde criamos uma função e dentro dela criamos um **"try"** e **"catch"** para filtrar os erros, e dentro do **"try"** chamamos a **"loginService"** que foi declarada no construtor herdando as funções da classe **"AppService"**, então chamamos a função que desejarmos localizadas dentro da service, e dentro dos parênteses passamos as informações pertinente a função que declaramos, em seguida o **".subscribe"** irá retornar a resposta da função, seja ela um erro ou não, nas variáveis **"resp"** e **"error"**.

3.7. Cors

O compartilhamento de recursos de origem cruzada (CORS) permite que solicitações de recursos de acesso de hosts remotos. Mostrarei como ativar o suporte CORS no Express.

Para efetuar a ativação do suporte CORS irei utilizar da aplicação proxy que criamos como modelo, para facilitar o entendimento em relação a correção de erros referente a CORS da aplicação node.js.

Para efetuar a configuração de CORS primeiro acesse a sua aplicação pelo VSC (Visual Studio Code), no caso acessei a proxy que irei usar como exemplo, em seguida abra a central de comando do VS e digite **"npm install cors"**, para instalar as dependências de CORS.

Em seguida abra a **"server.js"** onde fica a configuração referente ao acesso a app, após aberto crie uma variável que irá herdar as dependências de CORS, após a declaração vá abaixo da variável app e declare que app use a variável de cors, assim ela utilizará a dependência de cors, resolvendo erros referente a compartilhamento de origens cruzadas.

4. Padrões Angular

4.1. Padrões de desenvolvimento

Seção Padrões de Desenvolvimento	
Item	**Exemplo**
Nome variáveis sempre Camel Case	minhaVariável
Nome de funções sempre Camel Case	minhaFunção
Nome das classes Pascal Case	MinhaClasse
Nome das interfaces Pascal Case	MinhaModel
Efetuar chamadas externas pela Service	-
Alocar variáveis na module para transição de informações entre páginas	-
Utilizar nome de variáveis,funções e pastas o mais específico e claros possíveis.	-
Sempre que necessário usar routerlink para melhorar performace da aplicação	[routerlink] = "[´/´]"
Sempre verificar importação de componentes em app.module.ts, em cada alteração na aplicação	-
Centralizar css de toda a aplicação em uma pasta fora dos componentes	-
Sempre que possivel utilizar ngIf e ngfor para melhor desenpenho da aplicação	[ngIf]=""; , [ngfor]="";
Retorno de informação para proxy ou API´s apenas em formato JSON	-
Chamadas request feitas na service apenas em modelo websocket	Segue na Documentação "BOAS PRÁTICAS ANGULAR", item 5 , pag.13
Criar uma Service para cada tela da aplicação	-
Seção Padrões Criação de projeto	
Item	**Exemplo**
Criar aplicação através de comandos no prompt	ng new nome-da-aplicação --prefix=jad
Criar componentes através de comandos no prompt	ng g c nome-do-component --spec=false
Criar service através de comandos no prompt	ng g service nome-do-serviço
Criar module através de comando no prompt	ng g module nome-do-module

Verificar e sempre atualizar versão do Angular para a mais atual	(npm install -g @angular/cli) e (ng -v)
Verificar e sempre atualizar versão do Node para a mais atual	node -v
Verificar integridade de biblioteca antes msm de ser instalada	-
Utilização do servidor node para testes e desenvolvimento	ng serve
Criar maior número de componentes possível para evitar repetição de códigos	-
Ao iníciar um projeto utilizar apena um instalador de pacotes, preferencialmente NPM	npm, yarn, chocollate e etc.
Instalar diretórios de biblioteca node.js	npm install
Apagar arquivos criados .spec, pois não serão utilizados	-
Apagar arquivos criados .css de cada componente, pois serão centralizados em um arquivo separado	-
Seção Padrões Criação da proxy	
Item	**Exemplo**
Criar arquivo packge-lock.json	npm init
Craição de pastas através de comando	mkdir "nomedapasta"
Instalar diretórios de biblioteca node.js	npm install express body-parser morgan await uuid jsonwebtoken helmet config --save
Declarar bibliotecas como na documentação inicial de criação da proxy	Segue na Documentação "DESENVOLVIMENTO DE PROXY EM NODE.JS", item 2.1, pag.06
Utilizar a biblioteca Jwt para a criação do token de autenticação	Segue na Documentação "DESENVOLVIMENTO DE PROXY EM NODE.JS", item 2.1, pag.07
Utilizar a config.json para armazenamento de dados estáticos, que serão utilizados na proxy	Segue na Documentação "DESENVOLVIMENTO DE PROXY EM NODE.JS", item 2.3, pag.09
Todos retornos da proxy tem que ser em formato JSON	-
Utilizar por padrão o parâmetro process.env.PORT, como porta de execução da proxy	app.listen(process.env.PORT);
Não Utilizar POST para geração de Token	

4.2. Erros

Seção Aplicação		
Item	**Exemplo**	**Categoria**
Utilização de console.log(), em ambiente de HML e PRD	console.log()	
Utilização de console.log(), com variaveis sensiveis	console.log(numero_cartao)	Gr
Não utilização de proxy para comunicação com as API´s Bitpagg	return this.http.get(http://www.bitpagg.com.br/api/autenticador/Login/.concat('autenticador/ping'));	Gravis
Não utilizar service para comunicação com a proxy	-	
Não utilizar module para comunicação de variveis entre páginas	-	
Utilização de dados sensiveis como parametros fixos (hard code)	Codigo Campanha,Codigo Parceiro e etc.	G
Salvar em cache dados sensiveis do usuario	CPF, Dados do cartão e etc.	G
Arquivos de imagem, fontes e outros, criar pastas separadas para cada segmento	-	
Seção Proxy		
Item	**Exemplo**	**Categoria**
Não autenticação, ou autenticação incorreta das API´s	router.get('/autenticador/ping_auth', autenticador.PingAuth);	Grav
Padrão de chamada de API diferente do indicado pela documentação	Segue na Documentação "BOAS PRÁTICAS ANGULAR", item 5 , pag.13	
Não deverá conter em HML e PRD codigos referentes a liberação de CORS	-	
Não deverá estar setada para utilização a porta 3000 em HML e PRD	-	

Seção	
Item	**Descrição**
Leve	
Médio-Leve	
Médio	
Médio-Grave	
Grave	
Gravissimo	G

4.3. Comandos mais utilizados “npm”

Seção Npm	
Item	**Comando**
Verificar e atualizar os pacostes do projeto bem como rodar npm audity para vulnerabilidades, e adequar as vunerabilidades encontradas	npm audit
Verificar e atualizar bibliotecas node instaladas	npm install
Verificar e sempre atualizar versão do Angular para a mais atual	npm install -g @angular/cli
Para a criação de um arquivo.json (db.json) para consultas backend	npm install -g json-server
Alteração nas configurações no package-lock.json, para a criação do arquivo	npm init
Usado para a instalação do modulo node.js no projeto	npm install express
Instalar dependencias de CORS para o compartilhamento de recursos de origem cruzada	npm install cors
Biblioteca usada para verificação de conteudo que chega na aplicação através do req.body	npm install body-parse
Utiizada para otimizar a comunicação com as API´s	npm install morgan
Utilizada para a criação e manipulação de promises no projeto	npm install await
Utilizado para a geração de sequencias aleatorias	npm install uuid
Utilizado para retornar um web token para autenticação	npm install jsonwebtoken
A biblioteca helmet existem funções usadas para definição de cabeçalhos de resposta HTTP	npm install helmet