

TEXTO DE APOIO



AULA 6

Web Mobile

Professor Pedro Henrique Cacique Braga



Universidade Presbiteriana
Mackenzie





Sumário



CRIANDO O PRIMEIRO SERVIÇO BACK-END.....	3
TECNOLOGIAS DE BACK-END	5
INSTALANDO AS FERRAMENTAS.....	5
INSTALANDO O NEST.JS.....	7
CRIANDO O PRIMEIRO PROJETO.....	8
ESTRUTURA DO PROJETO	9
TESTANDO O PROJETO.....	14

CRIANDO O PRIMEIRO SERVIÇO BACK-END

Nas aulas passadas, discutimos sobre métodos de geração de ideias, ideação e prototipagem para o desenvolvimento de aplicativos. Vimos algumas das boas práticas de desenvolvimento do aplicativo do ponto de vista de interface e experiência do usuário (UI e UX). Neste contexto, trabalhamos com o desenvolvimento em linguagens para web, como HTML, CSS e JS.

Agora que já temos uma boa noção sobre o desenvolvimento do front-end, trabalharemos com os serviços de back-end. Entendemos aqui que um serviço de back-end é criado para desenvolver a lógica do sistema, que está por trás da interface. Ou seja, estamos trabalhando com a parte em que o usuário final não tem acesso direto.

Nós já trabalhamos com alguns serviços assim, como as APIs do COVID-19 e dos super-heróis, por exemplo. Nas arquiteturas de sistemas modernos, é muito comum termos esta divisão de front e back-end.

O principal motivo desta divisão é a descentralização do código e o estabelecimento de responsabilidades para serviços menores. O que implica em termos um sistema com maior escalabilidade e melhor manutenção.

Estamos fazendo uso de uma técnica bastante conhecida na programação que é o “dividir para conquistar”, isto é, estamos dividindo as funcionalidades do sistema em pacotes menores que podem ser desenvolvidos de forma independente.

Os webservices, ou serviços para web, podem ser chamados por meio de uma URL (uniform resource locator), que permite o acesso a um ou mais destes pacotes menores do sistema.

Tomemos um grande site como exemplo: o Facebook, que hoje tem milhões de usuários ao redor do mundo. Esta rede social começou com um site para navegadores comuns e hoje é um serviço multiplataforma, presente em smartphones, tablets, televisões e outros dispositivos inteligentes.

Imagine que seus criadores tivessem feito toda a lógica do app apenas no projeto de front. Ao migrar para um aplicativo nativo para iOS ou Android, por exemplo, teriam que refazer todo seu código na linguagem de cada plataforma.

Qual poderia ser uma solução viável? Separar as funcionalidades do sistema em serviços menores (cada um na linguagem que melhor se adapte à sua natureza) e criar uma forma de acesso para cada um deles.

Podemos analisar um destes serviços separadamente, como o serviço de Login. A lógica é simples: dado um nome de usuário e uma senha, verificar se os dados batem com o que se tem armazenado nos bancos de dados e retornar um valor booleano: true, caso os dados estejam corretos, e false, caso contrário.

Veja que a funcionalidade do serviço é pequena, mas, tendo milhões de usuários, pode ser requisitada várias vezes por segundo. Se criarmos este serviço e atribuirmos a ele uma URL, podemos chamá-lo independentemente do acesso aos demais dados do sistema.

Dessa forma, é possível usar o mesmo serviço tanto na versão para navegador quanto na versão para smartphones ou qualquer outra plataforma.

Agora imagine ainda que, depois de um tempo de uso do serviço, seus criadores resolveram adicionar uma proteção maior para o tráfego de informação, já que a rede social começou a ter uma importância maior e um uso cada vez mais intenso.

Como temos um serviço dedicado ao login, podemos adicionar os métodos de segurança, como criptografia dos dados, sem termos de alterar os outros módulos do sistema. Este é um exemplo da manutenção de código simplificada.

Mais ainda, se quisermos aumentar o número de servidores que armazenam as informações de usuários, podemos escalar o sistema de login, criando um serviço de roteamento da informação com base no nome de usuário, por exemplo.

Fazendo alterações como esta, todas as versões do sistema devem ter a mesma resposta de login, mas com serviço mais robusto e seguro.

TECNOLOGIAS DE BACK-END

Existem várias linguagens que podem ser utilizadas para este tipo de serviço. A maior parte delas está relacionada à análise e à manipulação de dados de um servidor.

Hoje em dia, as linguagens baseadas em Javascript têm se mostrado bastante eficientes para acesso e manipulação de dados e, ainda assim, criar uma conexão via protocolo HTTP.

Um dos frameworks que mais se destacam por dar funcionalidade ao Javascript do lado do servidor é o node.js.



Saiba mais sobre o node.js em <<https://nodejs.org/en/>>.

Até pouco tempo atrás, o Javascript era uma linguagem de front-end, interpretada apenas nos navegadores. Com o surgimento do node, porém, ela alcançou também os servidores, o que agradou muitos desenvolvedores web.

A partir daí foram surgindo outros frameworks para servidores, com o intuito de simplificar ainda mais os processos do node. O framework escolhido para esta aula é o nest.js, que é uma interface mais intuitiva para o node.



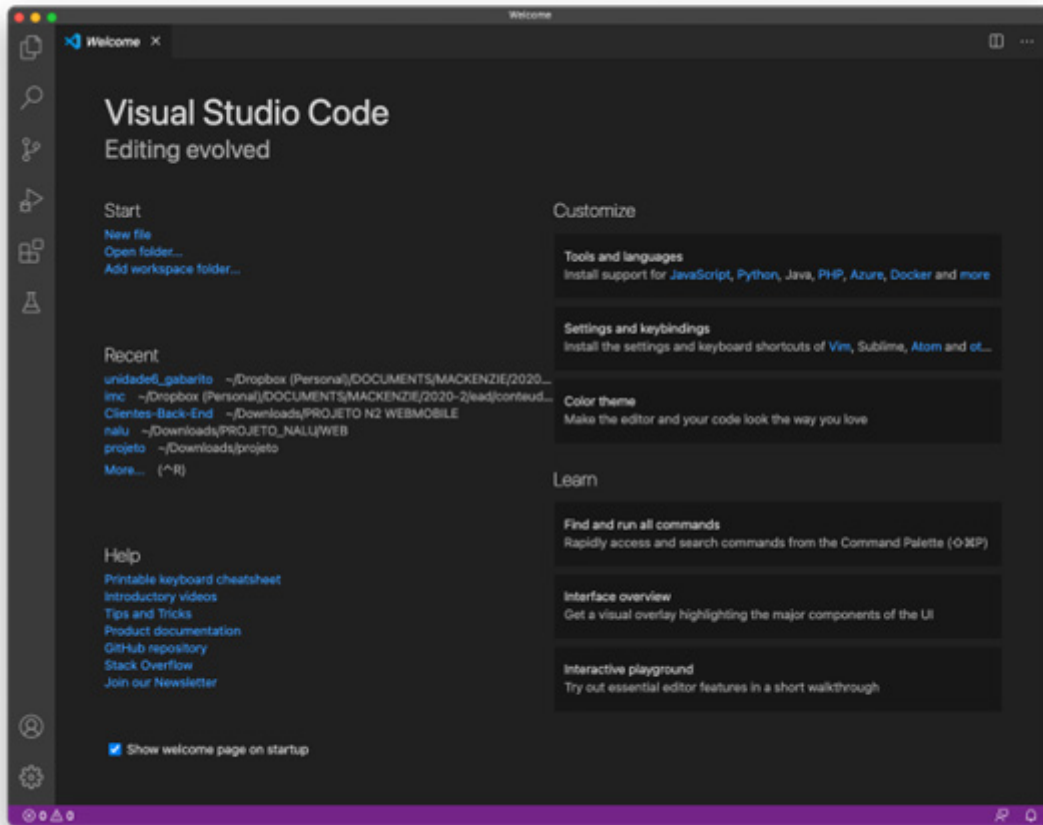
Saiba mais sobre o nest.js em <<https://nestjs.com>>.

INSTALANDO AS FERRAMENTAS

Para começarmos a criar um webservice, usaremos a interface de desenvolvimento (IDE) Visual Studio Code, ou VSCode, disponível para vários sistemas operacionais e para download gratuito em <<https://code.visualstudio.com>>.

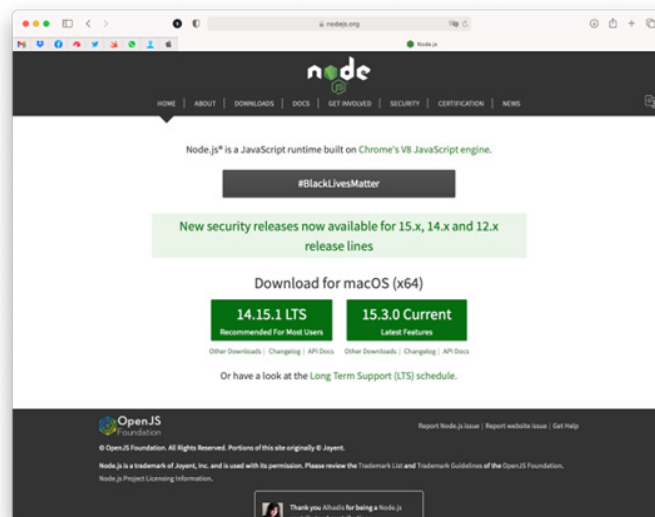
Trata-se de um editor de código bastante poderoso e, ao mesmo tempo, com interface simples e intuitiva, como mostra a Figura 1.

Figura 1 – VSCode



Além disso, instalaremos o node.js. Acesse o site <<https://nodejs.org/en/>> e baixe uma cópia do framework. Em seguida, execute o arquivo para instalação. Aconselho o uso da versão LTS que é a última versão estável disponível, como mostra a Figura 2.

Figura 2 – Download do node.js

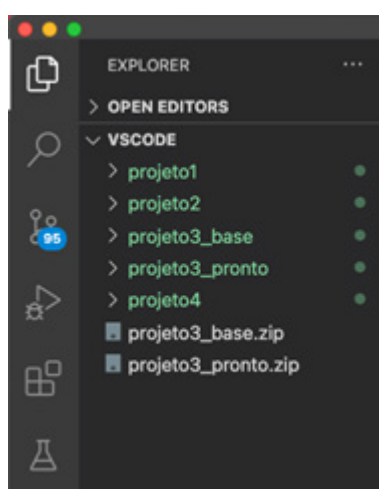


Por fim, faremos a instalação do pacote do Nest.js ao criarmos o projeto principal.

Abra o VSCode e, em seguida, selecione a opção Arquivo > Abrir. Navegue até uma pasta de sua preferência, que será seu workspace para os projetos. Você pode criar uma pasta em qualquer diretório e apontar para ela neste passo.

Ao abrir a pasta, você perceberá que a barra lateral esquerda mostrará todos os projetos contidos nela, como na Figura 3.

Figura 3 – Projetos no VSCode



Caso você tenha criado a pasta no passo anterior, aparecerá uma lista vazia.

INSTALANDO O NEST.JS

Agora, instalaremos o pacote de serviços do Nest.js. Para isso, abra o terminal no VSCode no menu Terminal > Novo Terminal. Em seguida, rode o seguinte comando:

```
npm i -g @nestjs/cli
```



Perceba que, no terminal, os comandos são executados após o símbolo %. Antes dele temos o nome de usuário e máquina seguido da pasta para o qual o terminal está apontando.

Neste caso, meu nome de usuário é pedrocacique, na máquina chamada Loki e a pasta criada para o workspace se chama vscode.

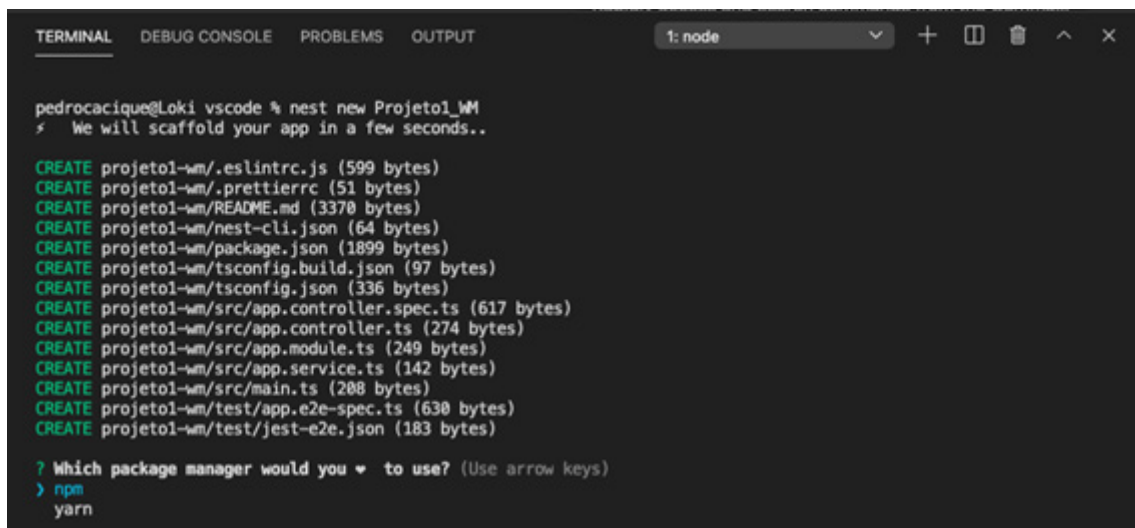
Este comando deve ser executado apenas uma vez por máquina em que estiver trabalhando.

CRIANDO O PRIMEIRO PROJETO

Agora que já estamos com as ferramentas instaladas, criaremos o primeiro projeto. Ainda no terminal aberto no VSCode, entre com o seguinte comando:

```
nest new <nome_do_projeto>
```

Substitua <nome_do_projeto> pelo nome adequado do projeto a ser criado.



```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT  1: node

pedrocacique@Loki vscode % nest new Projeto1_WM
$ We will scaffold your app in a few seconds..

CREATE projeto1-wm/.eslintrc.js (599 bytes)
CREATE projeto1-wm/.prettierrc (51 bytes)
CREATE projeto1-wm/README.md (3370 bytes)
CREATE projeto1-wm/nest-cli.json (64 bytes)
CREATE projeto1-wm/package.json (1899 bytes)
CREATE projeto1-wm/tsconfig.build.json (97 bytes)
CREATE projeto1-wm/tsconfig.json (336 bytes)
CREATE projeto1-wm/src/app.controller.spec.ts (617 bytes)
CREATE projeto1-wm/src/app.controller.ts (274 bytes)
CREATE projeto1-wm/src/app.module.ts (249 bytes)
CREATE projeto1-wm/src/app.service.ts (142 bytes)
CREATE projeto1-wm/src/main.ts (208 bytes)
CREATE projeto1-wm/test/app.e2e-spec.ts (630 bytes)
CREATE projeto1-wm/test/jest-e2e.json (183 bytes)

? Which package manager would you like to use? (Use arrow keys)
> npm
  yarn
```

Veja que chamei nosso primeiro projeto de Projeto1_WM. Ao executar, o terminal te perguntará qual gerenciador de pacotes você deseja usar. No nosso caso, usaremos o npm (Node Package Manager), então basta pressionar enter para continuar com a criação do projeto.

Este passo pode demorar alguns segundos. Ao final, será apresentada a mensagem de sucesso e a sugestão de doação para o projeto que é de código aberto. Você pode doar se quiser, mas não é obrigatório.


```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT  1: zsh

pedrocacique@Loki vscode % nest new Projeto1_WM
$ We will scaffold your app in a few seconds..

CREATE projeto1-wm/.eslintrc.js (599 bytes)
CREATE projeto1-wm/.prettierrc (51 bytes)
CREATE projeto1-wm/README.md (3370 bytes)
CREATE projeto1-wm/nest-cli.json (64 bytes)
CREATE projeto1-wm/package.json (1899 bytes)
CREATE projeto1-wm/tsconfig.build.json (97 bytes)
CREATE projeto1-wm/tsconfig.json (336 bytes)
CREATE projeto1-wm/src/app.controller.spec.ts (617 bytes)
CREATE projeto1-wm/src/app.controller.ts (274 bytes)
CREATE projeto1-wm/src/app.module.ts (249 bytes)
CREATE projeto1-wm/src/app.service.ts (142 bytes)
CREATE projeto1-wm/src/main.ts (208 bytes)
CREATE projeto1-wm/test/app.e2e-spec.ts (630 bytes)
CREATE projeto1-wm/test/jest-e2e.json (183 bytes)

? Which package manager would you like to use? npm
✓ Installation in progress...

🎉 Successfully created project projeto1-wm
👉 Get started with the following commands:

$ cd projeto1-wm
$ npm run start

Thanks for installing Nest 🙌
Please consider donating to our open collective
to help us maintain this package.

👉 Donate: https://opencollective.com/nest

pedrocacique@Loki vscode %
```

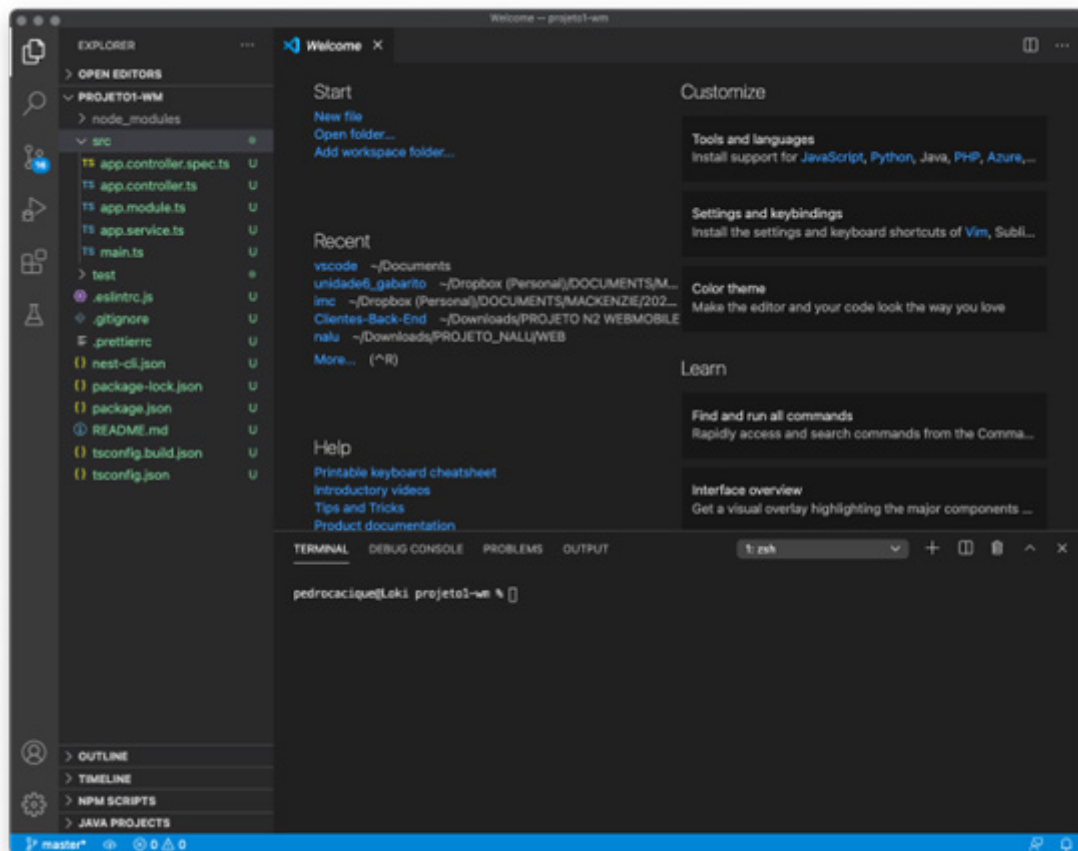
Agora que o projeto foi criado, abriremos sua pasta no VScode. Vá na opção Arquivo > Abrir e escolha a pasta criada para o projeto. Ao fazer isso, a tela do programa será recarregada, com os arquivos do projeto apenas, não mais com todos os projetos do workspace. Além disso, o terminal será atualizado para ser redirecionado a esta pasta.

Abra novamente o terminal (Terminal > Novo terminal).

ESTRUTURA DO PROJETO

Veja, na Figura 4, que o projeto apresenta uma estrutura de pastas e arquivos bem definida. Essa estrutura pode ser vista na barra lateral esquerda, com os arquivos e as pastas em cor verde.

Figura 4 – Estrutura do Projeto



Na pasta raiz, temos os arquivos de configuração do node.js, como o principal deles package.json, que traz todas as dependências que o projeto precisa, inclusive os comandos utilizados pelo nest.js.

Os arquivos principais do projeto, que trazem a lógica do nosso sistema, estão na pasta src. Nós temos cinco arquivos por padrão. Todos os arquivos que terminam com a extensão .spec.ts são utilizados para teste de software, portanto, neste momento, focaremos apenas nos demais.

Sendo assim, temos o arquivo main.ts que apresenta as configurações básicas do projeto:

```
src > TS main.ts > ...
1  import { NestFactory } from '@nestjs/core';
2  import { AppModule } from './app.module';
3
4  async function bootstrap() {
5    const app = await NestFactory.create(AppModule);
6    await app.listen(3000);
7  }
8  bootstrap();
9
```

Veja que ele realiza as importações das bibliotecas a serem utilizadas e cria um serviço que será executado na porta de número 3000. Você pode alterar este valor, caso sua máquina esteja usando esta porta para algum outro serviço.

Por enquanto, não precisamos alterar nada neste arquivo além do número da porta.

Os outros três arquivos representam as bases da estrutura do Nest.js:

- **Módulo:** responsável pelo redirecionamento de um serviço, importando todas as dependências necessárias para ele. Um projeto contém ao menos um módulo e pode ter vários outros, caso necessário para organização do serviço.
- **Controlador:** Responsável pelo mapeamento das rotas dos serviços oferecidos, ligando o método com a funcionalidade do serviço a uma URL específica, por meio dos verbos do protocolo HTTP.
- **Serviço:** responsável pela lógica do serviço em si.

Veja como o projeto está organizado então:

O arquivo `app.module.ts` contém as importações de bibliotecas no começo, depois a definição do módulo com a anotação `@Module`. Essa anotação traz três importantes propriedades:


1. Imports: uma lista de bibliotecas adicionais que o módulo utiliza.
2. Controllers: a lista de controladores do módulo.
3. Providers: a lista de providers com seus serviços.

Por fim, ele cria a instância do aplicativo (linha 10) para que seja executado.

```
src > TS app.module.ts > ...
1  import { Module } from '@nestjs/common';
2  import { AppController } from './app.controller';
3  import { AppService } from './app.service';
4
5  @Module({
6    imports: [],
7    controllers: [AppController],
8    providers: [AppService],
9  })
10 export class AppModule {}
11
```

O arquivo `app.controller.ts` contém o mapeamento das ações. Neste caso, temos um construtor que instancia um objeto relacionado ao serviço (que trará a lógica do módulo) e uma função inicial.

Esta função inicial está mapeada usando o método GET do protocolo HTTP. Veja que a assinatura do método é `getHello()`, o que indica a ausência de parâmetros iniciais e o método retorna um valor textual (string).



Acesse <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods> para saber mais sobre os métodos do protocolo HTTP.

```
src > TS app.controller.ts > ...
1  import { Controller, Get } from '@nestjs/common';
2  import { AppService } from './app.service';
3
4  @Controller()
5  export class AppController {
6      constructor(private readonly appService: AppService) {}
7
8      @Get()
9      getHello(): string {
10         return this.appService.getHello();
11     }
12 }
13
```

O método em si faz a chamada de outra função com a lógica do serviço, que está dentro da classe `AppService`. Veja que, na linha 10, a variável `appService`, criada no construtor é chamada para a invocação do método `getHello()` dentro dela.

Perceba que temos dois métodos diferentes com o mesmo nome, o primeiro está na classe `controller` e chama o segundo, que está na classe `service`.

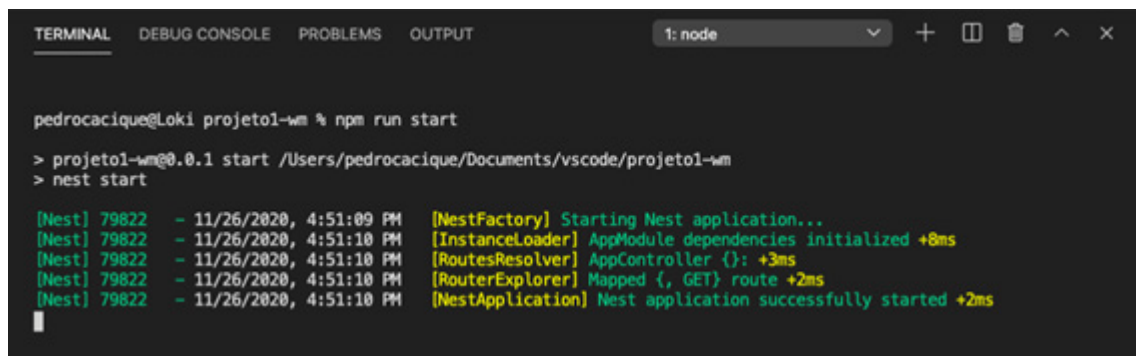
Por fim, a classe `app.service.ts` contém a lógica do nosso serviço que, neste caso, trata-se apenas da criação da frase “Hello World!”.

```
src > TS app.service.ts > ...
1  import { Injectable } from '@nestjs/common';
2
3  @Injectable()
4  export class AppService {
5      getHello(): string {
6          return 'Hello World!';
7      }
8  }
9
```

Rodando o projeto

Para rodarmos o projeto e colocarmos o serviço online, usamos o comando a seguir no terminal do VSCode:

`npm run start`



```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT  1: node
pedrocacique@Loki projet01 ~ % npm run start
> projet01 ~ % 0.0.1 start /Users/pedrocacique/Documents/vscode/projet01 ~ %
> nest start

[Nest] 79822 - 11/26/2020, 4:51:09 PM [NestFactory] Starting Nest application...
[Nest] 79822 - 11/26/2020, 4:51:10 PM [InstanceLoader] AppModule dependencies initialized +8ms
[Nest] 79822 - 11/26/2020, 4:51:10 PM [RoutesResolver] AppController {}: +3ms
[Nest] 79822 - 11/26/2020, 4:51:10 PM [RouterExplorer] Mapped {, GET} route +2ms
[Nest] 79822 - 11/26/2020, 4:51:10 PM [NestApplication] Nest application successfully started +2ms
```

Veja que o resultado foi positivo, apresentado pelas cinco linhas de resposta.

A leitura desta resposta nos mostra que:

1. A aplicação Nest foi executada.
2. As dependências foram carregadas no AppModule.
3. A classe `AppController` foi instanciada com sucesso.
4. Uma função com método GET foi mapeada na rota raiz.
5. A aplicação Nest foi inicializada com sucesso.

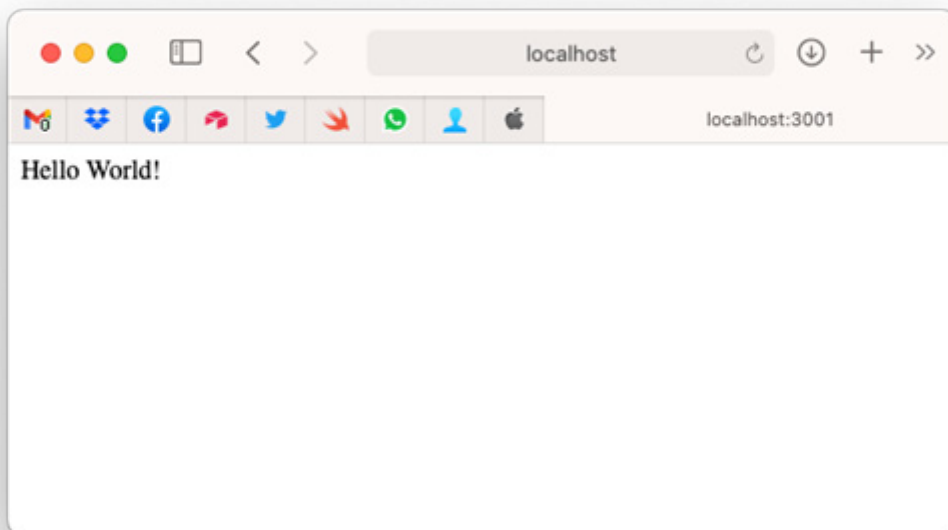
TESTANDO O PROJETO

Para testarmos, abra seu navegador de preferência e navegue para a URL do seu projeto, que está sendo hospedado no seu IP local, ou seja:

`http://localhost:3000/`

Você deve receber uma página com a frase Hello World! no corpo, como na Figura 5.

Figura 5 – Teste do Projeto



Muito bom!

Já temos um serviço funcional no back-end. Em nossa próxima aula, transformaremos este projeto em algo que possa ser utilizado para um aplicativo de controle de alunos de um curso.