SUMÁRIO

Introdução

Mongo, Express, Vue.js, and Node.js (**MEVN**)
MongoDB, Express, React, and Node.js (MERN)
MongoDB, Express, Angular, and Node.js (MEAN)

Vue.js

Vue.js é uma biblioteca (lib) javascript para o desenvolvimento de componentes reativos para interfaces web modernas.

npm

npm – Node Package Manager – cuida dos pacotes que nós instalamos para Node.js.

Lista de pacotes disponíveis: https://www.npmjs.com/

O package.json é responsável por "descrever" o seu projeto e informar as dependências de produção, url do repositório, versão em que está o seu projeto e outras informações.

npm install vue

```
senai-prj-workshops helena$ npm install vue
```

vue-cli facilita para realizar a criação de um novo projeto, para isto, vamos rodar o comando:

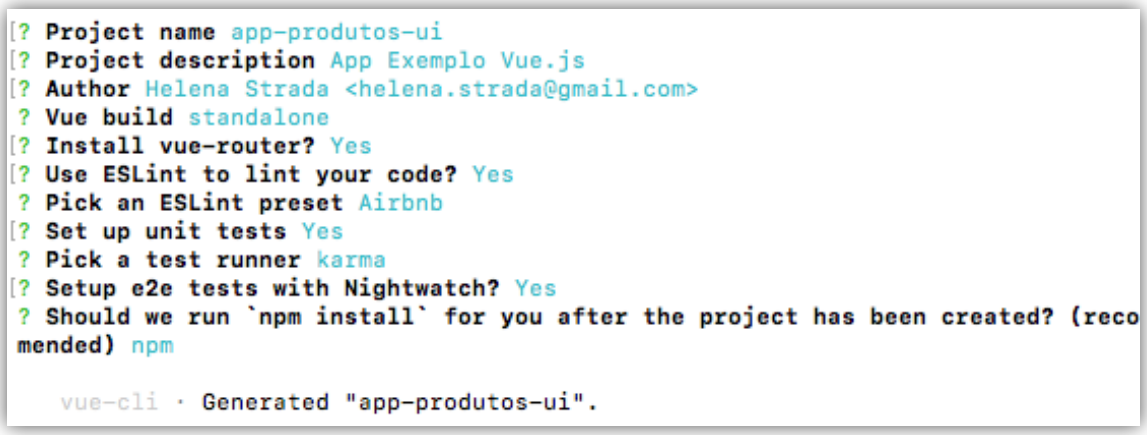
npm install –g vue-cli

Caso dê um erro de permissão negada, rode o terminal como administrador no Windows ou no MacOS e Linux:

```
helena$ sudo npm install -g vue-cli
```

Criando um novo projeto

vue init webpack app-produtos-ui



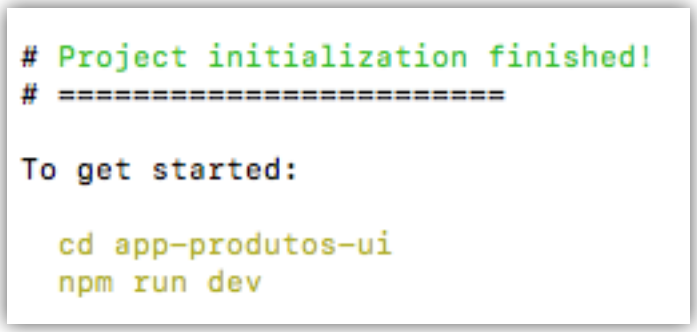
Opções de configuração

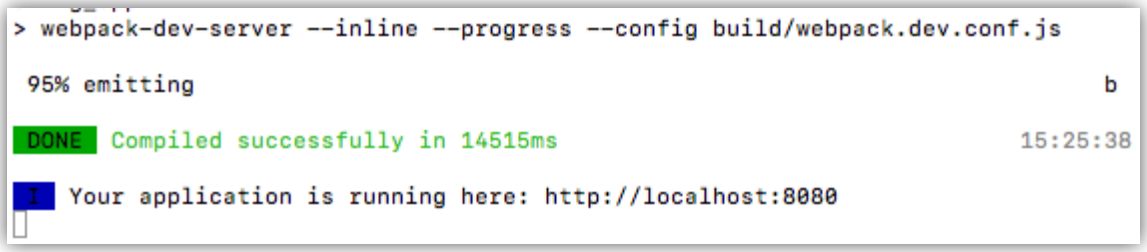
- **Vue build**: You will find two options to build the Vue.js app: runtime + compiler, or runtime Only. This has to do with the template compiler:
    - **Runtime only**: The runtime option is used to create the `vue` instances. This option does not include the template compiler.
    - **Runtime + compiler**: This option includes the template compiler, which means the `vue` templates are compiled to the plain JavaScript render functions.
- **Vue-router**: Vue-router is the official router for Vue.js applications. This option is specially used when we want to make our application a **Single Page Application** (**SPA**). When using this option, the application makes all the necessary requests one time when the page initially loads and sends requests to the server when new data is needed. We will be talking more about Single Page and Multi-Page applications in future chapters as well. For now, we will be using the Vue-router.
- **ESLint**: ESLint is a JavaScript linter tool. It is a static code analysis tool used to find the errors or the mistakes in the code. It basically makes sure that the code follows the standard guidelines. There are two options for choosing the ESLint from as well: standard linting or the Airbnb linting. We will be going with Airbnb for this project.
- **Setup test**: By setting up tests, the project creates a wrapper for the tests that we will be writing for our application. It creates the necessary structure and configuration for the tests codes to be able to be run. We will be using this option as well. For the test runner, we will be using Mocha and Karma, and for the end to end testing, we will be using Nightwatch, which we will learn about in further chapters.

- **Dependency management**: Lastly, to manage the packages and the dependencies, here we have two options: `npm` and `Yarn`. We mostly talked about `npm` in previous chapters. `Yarn` is also a dependency management tool just like `npm`. Both Yarn and `npm` have their own benefits, but for this application, we are going to use `npm`. You can learn more about Yarn here (https://yarnpkg.com/en/).

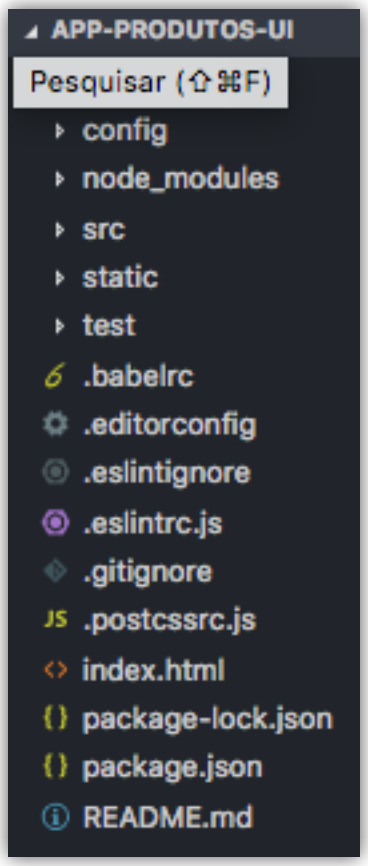
Rodando o projeto

$ cd app-produtos-ui
$ npm run dev





Estrutura de Pastas



- build folder: This folder contains the webpack configuration files for different environments: development, test, and production
- config folder: All the configurations of the application would go here
- node_modules: All the npm packages that we install reside in this folder
- src: This folder contains all the files related to rendering the components in the browser:
    - assets: You can add your CSS and images for your application inside this folder.
    - components: This folder will house all the frontend rendering files that will have a .vue extension.
    - router: This folder will take care of all the URL routes for different pages throughout the application.
    - App.vue: You can think of App.vue as the main component for rendering the view files. Other files will extend the layout defined on this file to create different views.
    - main.js: This is the main entry point for any Vue.js application.
- Static: You can use this folder as well to keep your static files, such as CSS and images.
- Test: This folder will be used to handle all the tests written for our application.

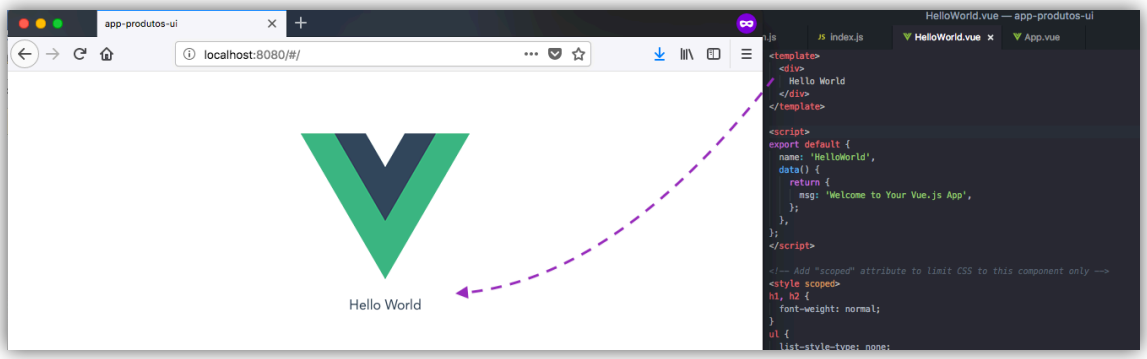
webpack e main.js

Dentro de build/webpack.base.conf.js, tem uma seção de module.exports:

Isso indica que o ponto de entrada da sua aplicação será o arquivo main.js localizado dentro da pasta src.

Redefinindo a página principal

Vamos alterar o componente HelloWorld.vue. Vamos remover todo o conteúdo dentro da div e apenas escrever Hello World.



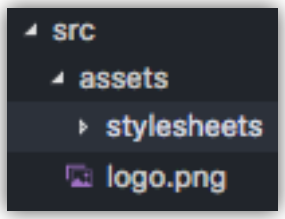
Após realizarmos as alterações, vamos rodar o projeto e verificar suas mudanças.

CSS

Trabalhando com estilos

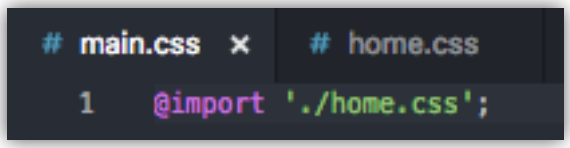
Para organizar os nossos estilos, vamos montar uma nova estrutura.

Dentro da pasta de src/assets, vamos criar uma nova pasta chamada ‘stylesheets’.

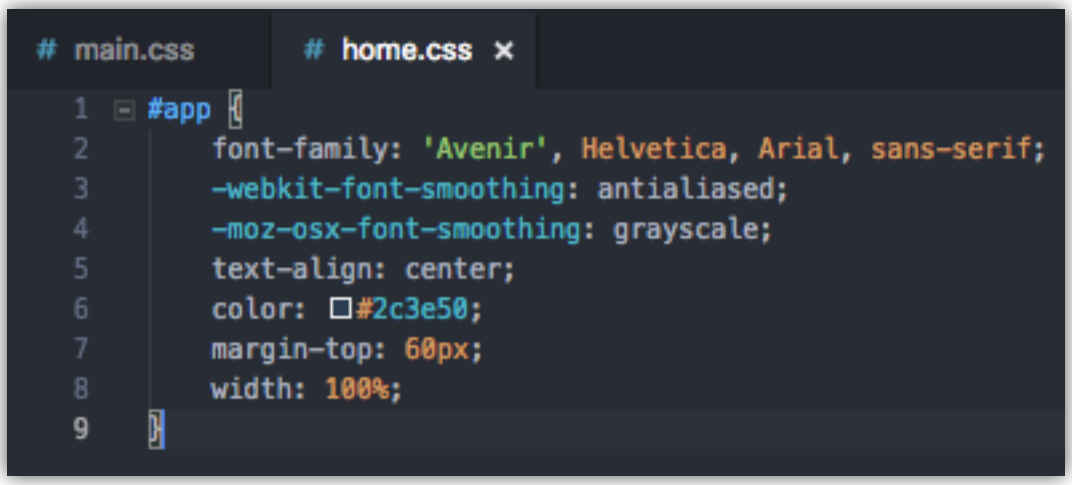


E dentro dessa pasta, vamos criar dois novos arquivos: *main.css* e *home.css*.

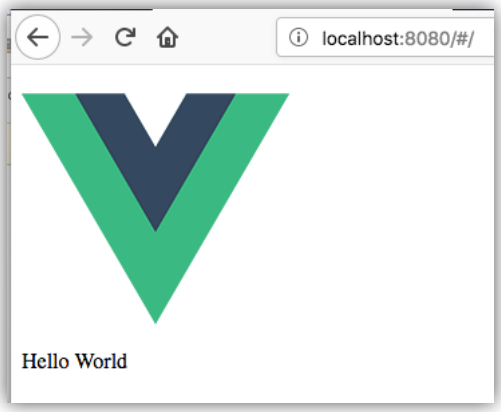
No arquivo main.css, vamos importar o arquivo home.css.



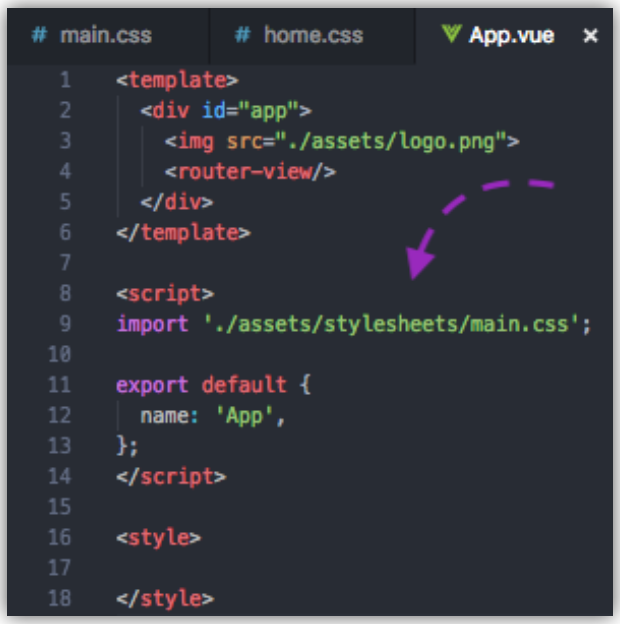
E no arquivo home.css, vamos recortar o css do App.vue e vamos colar neste arquivo.



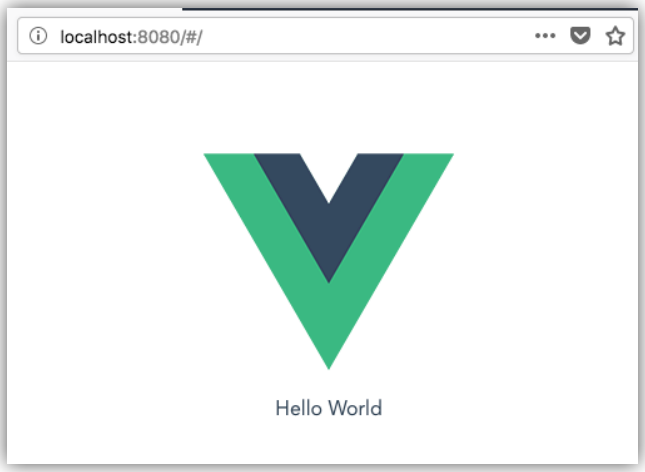
Se você estiver rodando o projeto, irá reparar que as propriedades foram perdidas. Por quê isso aconteceu sendo que criamos os arquivos?



O que acontece é que nós ainda não fizemos a referência para o arquivo main.css no nosso App.vue.



Agora, tudo voltou ao normal.



Dicas de bibliotecas

https://vuetifyjs.com
https://bootstrap-vue.js.org/docs

Um pouco sobre Vuetify

O Vuetify é um framework responsivo em Vue, baseado no Material Design. Contém detalhes minimalistas que auxiliam na construção dos componentes.

Vamos colocá-lo em nosso projeto juntamente com o bootstrap.

sudo npm install bootstrap --save
sudo npm install bootstrap-vue --save
sudo npm install vuetify --save

Vale lembrar que, até este momento, nós só trabalharmos com o layout e design das páginas.

Possíveis erros

Caso dê algum erro nesse início, coloque a versão do bootstrap no arquivo package.json para a que segue:

E rode o comando npm install novamente no cmd, na pasta do projeto.

Incluindo as bibliotecas

No arquivo src/main.js, vamos adicionar estes pacotes em nosso arquivo. São os mesmos pacotes que acabamos de incluir em nosso projeto.

```
import Vuetify from 'vuetify';

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap-vue/dist/bootstrap-vue.css';
import BootstrapVue from 'bootstrap-vue';

import Vue from 'vue';
import App from './App';
import router from './router';

Vue.use(BootstrapVue);
Vue.use(Vuetify);

Vue.config.productionTip = false;

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>',
});
```

No arquivo index.html vamos adicionar duas linhas. A primeira referente aos estilos que o vuetify possui e a outra são as fontes que o mesmo utiliza.

```
<link href="https://unpkg.com/vuetify/dist/vuetify.min.css" rel="stylesheet">
<link
href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700|Material+
Icons" rel="stylesheet">
```

Alterando nosso App.vue

Vamos alterar a apresentação em App.vue.

```
<template>
  <v-app id="inspire">
    <v-navigation-drawer
      fixed
      v-model="drawer"
      app
    >
      <v-list dense>
        <router-link v-bind:to="{ name: 'HelloWorld' }" class="side_bar_link">
          <v-list-tile>
            <v-list-tile-action>
              <v-icon>home</v-icon>
            </v-list-tile-action>
            <v-list-tile-content>Home</v-list-tile-content>
          </v-list-tile>
        </router-link>
        <router-link v-bind:to="{ name: 'Produtos' }" class="side_bar_link">
          <v-list-tile>
            <v-list-tile-action>
              <v-icon>contact_mail</v-icon>
            </v-list-tile-action>
            <v-list-tile-content>Produto</v-list-tile-content>
          </v-list-tile>
        </router-link>
      </v-list>
    </v-navigation-drawer>
    <v-toolbar color="indigo" dark fixed app>
      <v-toolbar-side-icon @click.stop="drawer = !drawer"></v-toolbar-side-
icon>
      <v-toolbar-title>Home</v-toolbar-title>
    </v-toolbar>
    <v-content>
      <v-container fluid>
        <div id="app">
          <router-view/>
        </div>
      </v-container>
    </v-content>
    <v-footer color="indigo" app>
      <span class="white--text">&copy; 2018</span>
    </v-footer>
```

```
  </v-app>
</template>
```

Alterando nossa página principal

Vamos alterar o estilo em home.css.

```
#app {
    font-family: 'Avenir', Helvetica, Arial, sans-serif;
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
    text-align: center;
    color: #2c3e50;
}

#inspire {
    font-family: 'Avenir', Helvetica, Arial, sans-serif;
}

.container.fill-height {
    align-items: normal;
}

a.side_bar_link {
    text-decoration: none;
}
```

Alterando o componente src/components/HelloWorld.vue.

```
<template>
  <v-layout>
    this is home
  </v-layout>
</template>
```

Criando o componente de produto

Vamos criar o novo componente para a página de produtos em src/components chamado Produtos.vue.

```
<template>
  <v-layout>
    página de produtos
  </v-layout>
</template>
```

No arquivo src/router/index.js, vamos incluir o novo componente criado e adicionar uma nova rota.

```
import Vue from 'vue';
import Router from 'vue-router';
import HelloWorld from '@/components/HelloWorld';
import Produtos from '@/components/Produtos';

Vue.use(Router);

export default new Router({
  routes: [
    {
      path: '/',
      name: 'HelloWorld',
      component: HelloWorld,
    },
    {
      path: '/produtos',
      name: 'Produtos',
      component: Produtos,
    },
  ],
});
```

Após incluirmos essa nova rota, vamos realizar o teste de nossa aplicação.

Apresentando os produtos fixos em nossa página principal

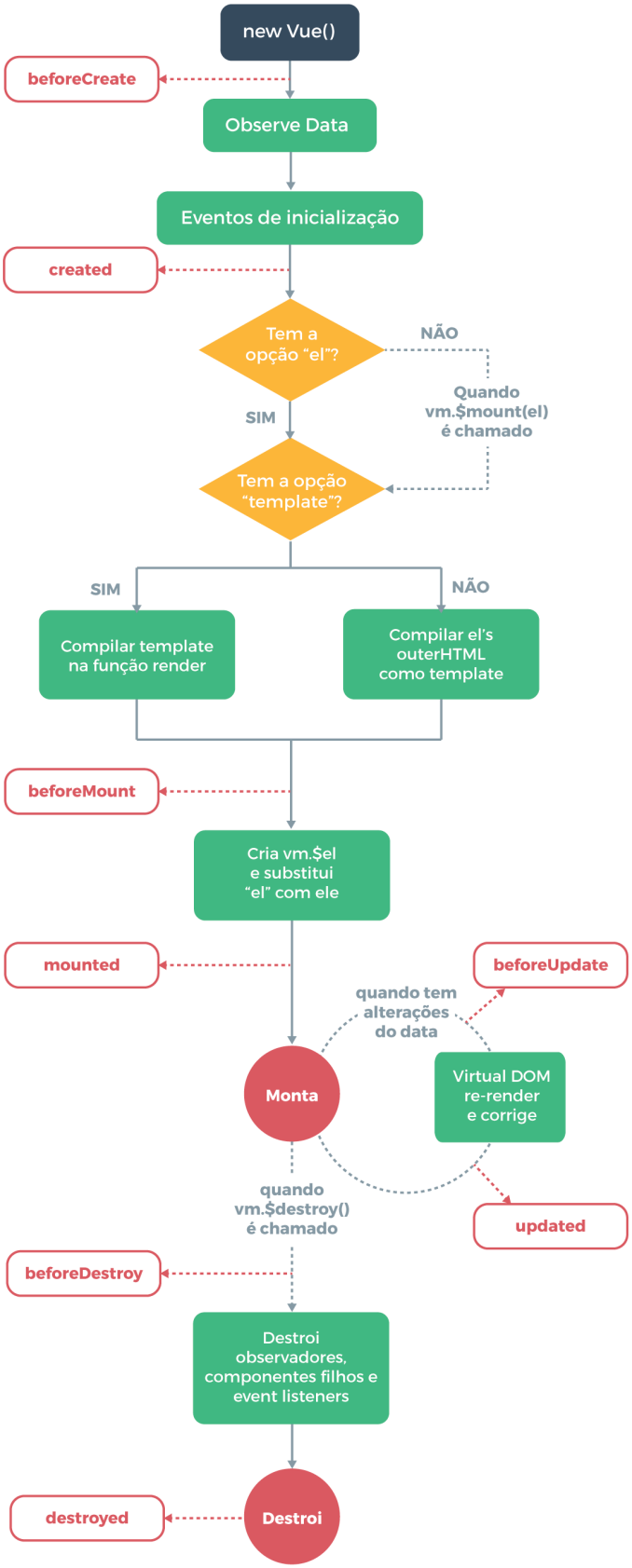
Vamos alterar o componente HelloWorld para apresentar a lista de produtos.

```
<template>
  <v-layout row wrap>
    <v-flex xs4>
      <v-card>
        <v-card-title primary-title>
          <div>
            <div class="headline">Nome Produto A</div>
            <span class="grey--text">01/01/2018</span>
          </div>
        </v-card-title>
        <v-card-text>
          Descrição Produto A
        </v-card-text>
        <v-card-actions>
          <v-btn flat color="purple">Ver informações</v-btn>
          <v-spacer></v-spacer>
        </v-card-actions>
      </v-card>
    </v-flex>
```

```
<v-flex xs4>
  <v-card>
    <v-card-title primary-title>
      <div>
        <div class="headline">Nome Produto B</div>
        <span class="grey--text">01/01/2018</span>
      </div>
    </v-card-title>
    <v-card-text>
      Descrição Produto B
    </v-card-text>
    <v-card-actions>
      <v-btn flat color="purple">Ver informações</v-btn>
      <v-spacer></v-spacer>
    </v-card-actions>
  </v-card>
</v-flex>
<v-flex xs4>
  <v-card>
    <v-card-title primary-title>
      <div>
        <div class="headline">Nome Produto C</div>
        <span class="grey--text">01/01/2018</span>
      </div>
    </v-card-title>
    <v-card-text>
      Descrição Produto C
    </v-card-text>
    <v-card-actions>
      <v-btn flat color="purple">Ver informações</v-btn>
      <v-spacer></v-spacer>
    </v-card-actions>
  </v-card>
</v-flex>
<v-flex xs4>
  <v-card>
    <v-card-title primary-title>
      <div>
        <div class="headline">Nome Produto D</div>
        <span class="grey--text">01/01/2018</span>
      </div>
    </v-card-title>
    <v-card-text>
      Descrição Produto D
    </v-card-text>
    <v-card-actions>
      <v-btn flat color="purple">Ver informações</v-btn>
      <v-spacer></v-spacer>
    </v-card-actions>
  </v-card>
</v-flex>
```

```
  </v-layout>
</template>
```

Ciclo de vida



Listando conteúdo dinâmico

Até aqui, nós adicionamos um conteúdo fixo em nossa página principal. Na próxima etapa, vamos listar o conteúdo dinâmico e fazer a integração entre front-end e back-end.

Adicionando o axios

Vamos utilizar o *axios* para fazer a comunicação entre o nosso cliente e o servidor. Para isto, instale na pasta onde o seu projeto está localizado:

npm install axios --save

Realizando a chamada à API

Vamos continuar alterando o arquivo HelloWorld.vue.

```
<script>
import axios from 'axios';

export default {
  name: 'HelloWorld',
  data() {
    return {
      produtos: [],
    };
  },
  mounted() {
    this.fetchProdutos();
  },
  methods: {
    async fetchProdutos() {
      return axios({
        method: 'get',
        url: 'http://localhost:8085/produtos',
      })
        .then((response) => {
          this.produtos = response.data;
        })
        .catch(() => {
        });
    },
  },
};
</script>
```

Alterando a apresentação dos produtos

HelloWorld.vue.

```
<template>
  <v-layout row wrap>
    <v-flex xs4 v-for="p in produtos" :key="p.id">
      <v-card>
        <v-card-title primary-title>
          <div>
            <div class="headline">{{ p.descricao }}</div>
            <span class="grey--text">{{ p.dataCadastro }}</span>
          </div>
        </v-card-title>
        <v-card-text>
          {{ p.urlImagem }}
        </v-card-text>
      </v-card>
    </v-flex>
  </v-layout>
</template>
```

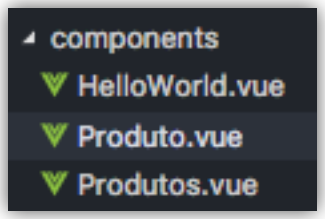
Visualizando apenas um produto

Vamos editar a home para que duas alterações sejam feitas. Um link externo seja criado, para a url da imagem, e o outro para que ao clicar em um determinado produto, apenas apareça suas próprias informações.

No arquivo HelloWorld.vue.

```
<template>
  <v-layout row wrap>
    <v-flex xs4 v-for="p in produtos" :key="p.id">
      <v-card>
        <v-card-title primary-title>
          <div>
            <div class="headline">
              <v-btn flat v-bind:to="`/produtos/${p.id}`">
                {{ p.descricao }}
              </v-btn>
            </div>
            <span class="grey--text">{{ p.dataCadastro }}</span>
          </div>
        </v-card-title>
        <v-card-text>
          <v-btn outline color="indigo" v-bind:to="`/produtos/${p.id}`">{{
p.urlImagem }}</v-btn>
        </v-card-text>
      </v-card>
    </v-flex>
  </v-layout>
</template>
```

Vamos criar um novo componente chamado Produto.



No código deste componente, vamos colocar as seguintes informações:

```
<template>
  <v-layout row wrap>
    <v-flex xs4>
      <v-card>
        <v-card-title primary-title>
          <div>
```

```
            <div class="headline">{{ produto.descricao }}</div>
            <span class="grey--text">{{ produto.dataCadastro }}</span>
          </div>
        </v-card-title>
        <v-card-text>
          {{ produto.urlImagem }}
        </v-card-text>
      </v-card>
    </v-flex>
  </v-layout>
</template>
<script>
import axios from 'axios';

export default {
  name: 'Produto',
  data() {
    return {
      produto: [],
    };
  },
  mounted() {
    this.fetchProduto();
  },
  methods: {
    async fetchProduto() {
      return axios({
        method: 'get',
        url: `http://localhost:8085/produtos/${this.$route.params.id}`,
      })
        .then((response) => {
          this.produto = response.data;
        })
        .catch(() => {
        });
    },
  },
};
</script>
```

Fazer apenas isto, não irá funcionar. Veremos apenas uma página em branco. Falta, em nossas rotas, identificar que ao irmos para esta rota, algum componente irá atender a sua chamada.

```
{
      path: '/produtos/:id',
      name: 'Produto',
      component: Produto,
    },
```

Código completo do index.js.

```
import Vue from 'vue';
import Router from 'vue-router';
import HelloWorld from '@/components/HelloWorld';
import Produtos from '@/components/Produtos';
import Produto from '@/components/Produto';

Vue.use(Router);

export default new Router({
  routes: [
    {
      path: '/',
      name: 'HelloWorld',
      component: HelloWorld,
    },
    {
      path: '/produtos',
      name: 'Produtos',
      component: Produtos,
    },
    {
      path: '/produtos/:id',
      name: 'Produto',
      component: Produto,
    },
  ],
});
```

vuex