Vue.js

Jadson Santos
Computing Engineer

Vue.js

- Vue (pronounced /vjuː/, like **view**) is a **progressive framework** for building user interfaces
- The core library is focused on the view layer only, and is easy to pick up and integrate with other libraries or existing projects.
- On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications

Vue.js

- Progressive framework is a framework that you can insert into your project as you feel the need for it
- Differently of other JavaScript framework like Angular, that since the beginning, you need a full project make in Angular, follow the “Angular rules”.
- This implies you need to learn a lot of things to start programming with Angular.

Vue.js

- Vue is more simple and flexible
- Vue allows you make just specific parts of your application. You learn just what is necessary for the problem you are dealing with.
- Or if it is necessary and you have time, you can learn more and make a full complex front-end application 100% Vue.

Vue.js

Vue.js

- **Vue** uses Javascript, **Angular** relies on TypeScript
- **Vue** is pretty easy to learn, **Angular's** learning curve is much steeper
- **Vue** Is indicated if you are working alone or have a small team, **Angular** is for really large applications.

Vue.js

- While **Angular** offering data binding to deal with the DOM without having to touch it directly, there were still extra concepts that demanded code be structured a certain way.

- **Vue** had these efficiencies, but was even more lightweight.

Vue.js Hello World

Vue.js Hello World

- Download it from:
    - https://vuejs.org/v2/guide/installation.html
- And Include the vue script in your HTML page

Vue.js Hello World

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello World - Vue.js</title>
    <script type="text/javascript" src="js/vue.js"></script>
</head>
<body>

</body>
</html>
```

Vue.js Hello World

- Now we have to indicate the part of HTML that Vue.js will look at and use.
- We will create the div with “app” id

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Hello World - Vue.js</title>
   <script type="text/javascript" src="js/vue.js"></script>
</head>
<body>
    <div id="app">   </div>
</body>
</html>
```

Vue.js Hello World

- Create a **new Vue()** object with the "**el**" and "**data**" elements.
  - **el** tell which element of our HTML we want Vue manipulate
  - The **"data"** field receives a javascript object with the data that will be manipulated in the context of our application

```
<script type="text/javascript">
    new Vue({
        el: "#app",
        data: {
            message: "Hello World!"
        }
    });
</script>
```

Vue.js Hello World

- To show the data of the variable "message" in the div app we use the **interpolation** putting the variable "message" between double keys {{ }} inside the div.

```
<body>
    <div id="app"> {{ message }}  </div>
    <script type="text/javascript">
        new Vue({
            el: "#app",
            data: {
                message: "Hello World!"
            }
        });
    </script>
</body>
```

Vue.js Hello World

- Open the html file in a browser

Hello World!

Inspe Cons Debu Style E Perforn Mem Netw Stora

Filter output Persist Logs

```
You are running Vue in development mode.                vue.js:8436:5
Make sure to turn on production mode when deploying for
production.
See more tips at https://vuejs.org/guide/deployment.html
```

Vue.js Hello World

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Hello World - Vue.js</title>
   <script type="text/javascript" src="js/vue.js"></script>
</head>
<body>
    <div id="app"> {{ message }}  </div>
    <script type="text/javascript">
        new Vue({
            el: "#app",
            data: {
                message: "Hello World!"
            }
        });
    </script>
</body>
</html>
```

VueJS Directives

Vue.js Directives

- Directives are the part of Vue.js that attach special meaning and behavior to plain **html elements** on the page

Vue.js V-model

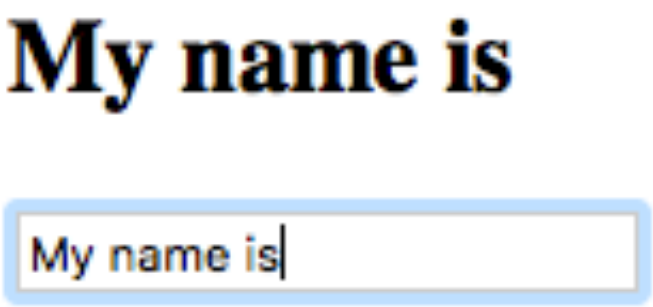
- **v-model** directive to make **two way binding** between the **HTML element** and the data attribute it refers to in the **Vue JavaScript** code

```
var app1 = new Vue({
    el: '#app-1',
    data: {
        message: 'Enter Your Name...'
    }
 })
```

```
<div id="app-1">
  <h2>{{ message }}</h2>
  <input v-model="message" class="form-control">
</div>
```

Vue.js V-model

- When you type your name in the input text with v-model, the value type if update in the message data of Vue model and the HTML view {{message}} is updated too.
  - Vue message variable ←→ HTML {{message}} interpolation



Vue.js Conditionals and Loops

- **v-if** directive can be used to test some conditions over the JavaScript object data
- **v-else** directive can be used when the conditions on the v-if fails
- **v-else-if** directive can be used much like you would use else-if clauses in JavaScript

Vue.js Conditionals and Loops

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    conditional: 2
  }
})
```

```
<div id="app-2">
  <span v-if="conditional == 0 "> Test Conditional v-if {{conditional}}</span>
  <span v-else-if="conditional == 1"> Test Conditional v-else-if {{conditional}}</span>
  <span v-else> Test Conditional v-else {{conditional}}</span>
</div>
```

Test Conditional v-else 2

Vue.js Directives Summary

- **v-model:** used to create a two way binding between the element and the data
- **v-show**: used to conditionally display an element (css based)
- **v-if, v-else, v-else-if**: used to conditionally render an element
- **v-for**: iterate over a range, an array, an object, or an array of objects

Vue.js Conditionals and Loops

- **v-for** directive can be used for interact over a list of items over the JavaScript object data

```
var app3 = new Vue({
  el: '#app-3',
  data: {
    linguagens: [
      { text: 'JavaScript' },
      { text: 'Jquery' },
      { text: 'Vue' }
    ]
  }
})
```

```
<div id="app-3">
  <ol>
    <li v-for="l in linguagens">
      {{ l.text }}
    </li>
  </ol>
</div>
```

Vue.js Directives Summary

- **v-on**: listen to various DOM events (click, keyup, submit, etc..)
- **v-bind**: used to dynamically bind one or more attributes, or a component prop, to an expression
- **v-text**: same as {{ interpolation }} and updates the element's textContent

Vue.js Components

- Components are reusable structures with encapsulated functionalities. That is, elements that have html, css and javascript encapsulated and that can be reused either within the same project or even in others projects

Vue.js Components

- In Vue, a component is essentially a Vue instance with pre-defined options

```
<ol id="components-demo">
    <my-item
        v-for="carItem in myListOfCars"
        v-bind:car="carItem"
        v-bind:key="carItem.id">
    </my-item>
</ol>

<script type="text/javascript">

    // component definition
    Vue.component('my-item', {
        props: ['car'],
        template: '<li>This is a Vue.js component of {{car.name}} </li>'
    })

    // new Vue instance
    new Vue({
            el: '#components-demo',
            data: {
                myListOfCars: [
                    { id: 1, name: 'Ferrari' },
                    { id: 2, name: 'Mclaren' },
                    { id: 3, name: 'Mercedes' }
                ]
            }
     })

</script>
```

Vue.js Class and Style Bindings

- Vue provides special enhancements when **v-bind** is used with class and style

```
<style>
    .active{
        color: blue;
    }
</style>


<ol id="components-demo">

    <div v-bind:class="{ active: isActive }"> bind css class active </div>

    <div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"> bind css style </div>

</ol>



<script type="text/javascript">

    // new Vue instance
    new Vue({
            el: '#components-demo',
            data: {
                isActive: true,
                activeColor: 'red',
                fontSize: 30
            }
     })

</script>
```

Vue.js Event Handing

- We can use the **v-on** directive to listen to DOM events and run some JavaScript when they're triggered

```
<div id="example-1">
    <button v-on:click="counter += 1">Add 1</button>
    <p>The button above has been clicked {{ counter }} times.</p>
    <br/><br/>
    <button v-on:click="myMethod">Greet</button>
    <br/><br/>
    <button v-on:click="sayHello('Hello')">Say Hello</button>

</div>


<script type="text/javascript">
    // new Vue instance
    var example1 = new Vue({
      el: '#example-1',
      data: {
        counter: 0,
        name: 'Vue.js'
      },
      // define methods under the `methods` object
      methods: {
        myMethod: function (event) {
            // `this` inside methods points to the Vue instance
            alert('Hello ' + this.name + '!')
            // `event` is the native DOM event
            if (event) {
                alert(event.target.tagName)
            }
        },
        sayHello: function (message) {
            alert(message)
        }
      }
    })
</script>
```


VUE CLI

Vue.js CLI

- You can develop a complete **Single Page Application** with Vue using vue-cli

- Install it via the **node.js** package manager (npm)

Vue.js CLI

- What is Node.js and NPM?
  - **Node.js** is javascript development platform, the uses the google's engine V8. Basically, you cat execute javascript applications on server side (out of a web browser)
  - **NPM** is a package manager of Node.js, where you can manager the javascript dependences of your project (like maven ou gradle does in Java world).

  - Similar Angular, Vue.js is not made in Node.js and does not need Node.js to run, but use its package manger.

Vue.js CLI

- Download and install node.js from:

https://nodejs.org/en/

- It will automatically install npm package manager

Vue.js CLI

- After that, install vue command line interface tool with the commands:

- $sudo npm install -g vue-cli

```
jadson@jadson-pc:~$ vue --version
2.9.6
```

Vue.js CLI

- Creating a new vue project with vue-cli:
- $ vue init <*template-name*> <*project-name*>
- This command creates a project-name folder with a new vue project using a vue template.
- What is a vue template?
  - Vue.js does not define a default project structure for vue projects, so you have choose what will be your project structure.

Vue.js CLI

- There are three principal vue templates:
  - **webpack** – Uses the webpack as module bundler. It has support to vue-loader with hot reload, javascript lint and units tests. It is the **more complete** template.
  - **webpack-simple** – Simplification of webpack structure, for example, it do not creates the unit test folders
  - **browserify** - It has support to vue-loader with hot reload, javascript lint and units tests. It is **more simple** that webpack.

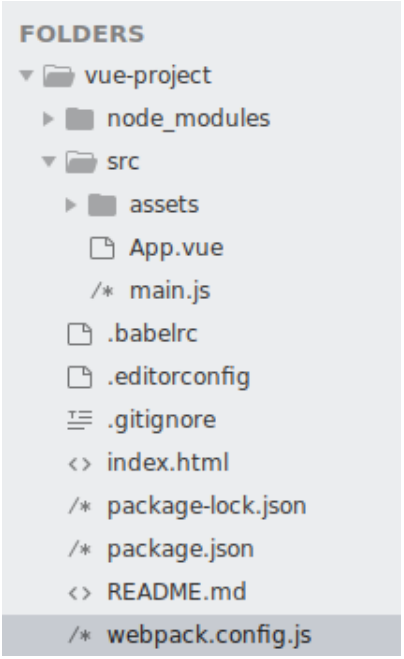
Vue.js CLI

- We will use the webpack, to to create a project called “hello-vue-cli”, use the command:

- $ vue init webpack-simple hello-vue-cli

SPA with Vue.js

- To run the empyt project:
  - cd hello-vue-cli
  - npm install (to downdoad the dependences)



node_modules: vue.js dependences

src: Our Vue code

assets: imgs and other external files of our application

App.vue: The main vue componet

index.html: main and unique page of application

SPA with Vue.js

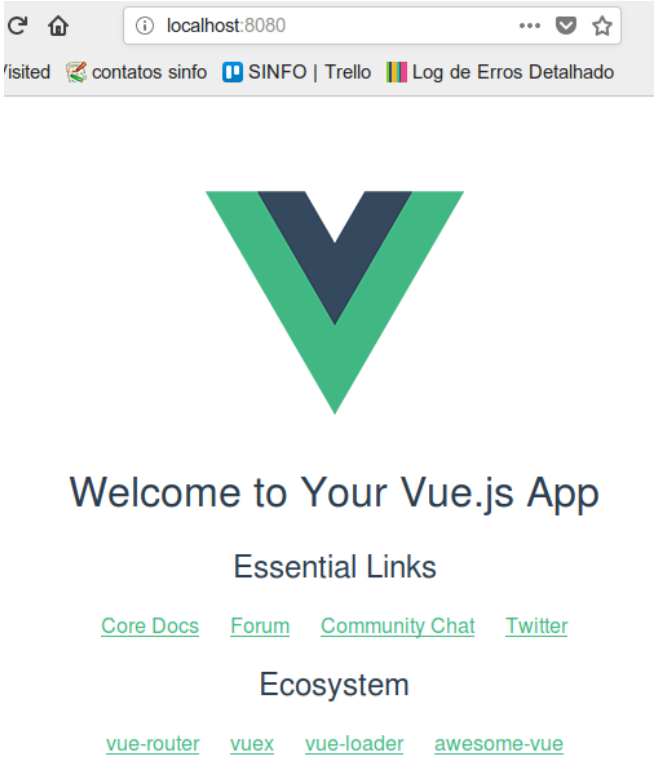
- The default structure of a vue component is composed of three elements: template, script, and style

App.vue × index.html × main.js ×

```
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <h1>{{ msg }}</h1>
    <h2>Essential Links</h2>
    <ul>
      <li><a href="http://router.vuejs.org/" target="_blank">vue-router</a></li>
    </ul>
  </div>
</template>

<script>
export default {
  name: 'app',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
```

SPA with Vue.js

- To run the empyt project
  - npm run dev (development environment)



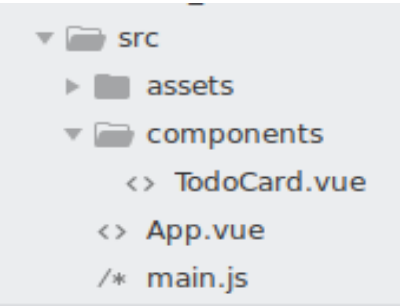
SPA with Vue.js

- Vue.js install dependences
  - npm install font-awesome -- save
  - npm install bulma -- save
  - npm install bootstrap -- save
  - etc…
- This will be install inside the node_modules. To include them, open tem index.html and include it on <head> tag

```
<head>
  <meta charset="utf-8">
  <title>vue-project</title>
  <link rel="stylesheet" href="./node_modules/bulma/css/bulma.css" type="text/css">
  <link rel="stylesheet" href="./node_modules/font-awesome/css/font-awesome.min.css" type="text/css">
</head>
```

Create a new Vue Component

- Create a new .vue file with template, script and css.
  - Give a name to this new component




```
<script>
  // the name of component, use as <todo-card> </todo-card>
  export default {
    name: 'todo-card'
  }
</script>
```

Create a new Vue Component

- Import it com App.vue component
  - Use this new component with the same name you declare insde componet .vue file

```
<script>

import TodoCard from './components/TodoCard'

export default {
  name: 'app',

  /// Define the component ///
  components: {
    TodoCard
  },

  data () {
    return {
      msg: 'Hello, World!'
    }
  }
}
</script>
```

```
-->
<div class="row">
  <div class="columns">
    <div class="column is-half is-offset-one-quarter">
      <todo-card></todo-card>
    </div>
  </div>
</div>
```

Vue.js Tools

Visual Studio Code

- Syntax Highlight
- Compiler
- Integrated Terminal

routes.js - vue-routers-prerender - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

EXPLORER

OPEN EDITORS 1 UNSAVED

routes.js src 2, U

VUE-ROUTERS-PRERENDER

dist

node_modules

src

assets

components

App.vue

main.js

routes.js 2, U

.babelrc

.editorconfig

.gitignore

index.html

package-lock.json

package.json

README.md

webpack.config.js

OUTLINE

```
import Bar from './components/Bar'
import Foo from './components/Foo'

import User from './components/Users/User'
import UserProfile from './components/Users/Profile'
import UserPosts from './components/Users/Posts'


// 2. Define some routes
// Each route should map to a component. The "component" can
// either be an actual component constructor created via
// `Vue.extend()`, or just a component options object.
// We'll talk about nested routes later.
const routes = [
  { path: '/foo', component: Foo },
  { path: '/bar, component: Bar },
  { path: '/user/:id', component: User,
    children: [
      {
        // UserProfile will be rendered inside User's <router-view>
        // when /user/:id/profile is matched
        path: 'profile',
        component: UserProfile
      },
      {
        // UserPosts will be rendered inside User's <router-view>
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

Filter. Eg: text, **/*.ts, !**/node_modules/**

routes.js src 2

[ts] Unterminated string literal. (18, 34)

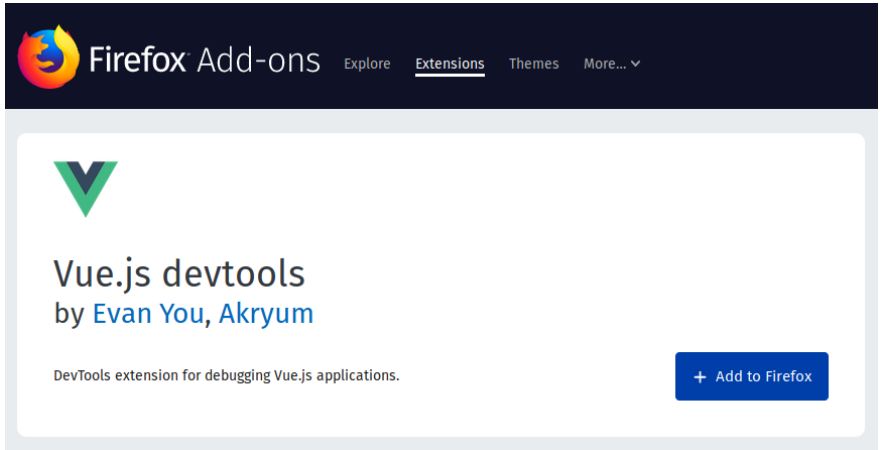
[ts] ',' expected. (19, 2)

master* 2 0

Ln 22, Col 20 Tab Size: 2 UTF-8 LF JavaScript

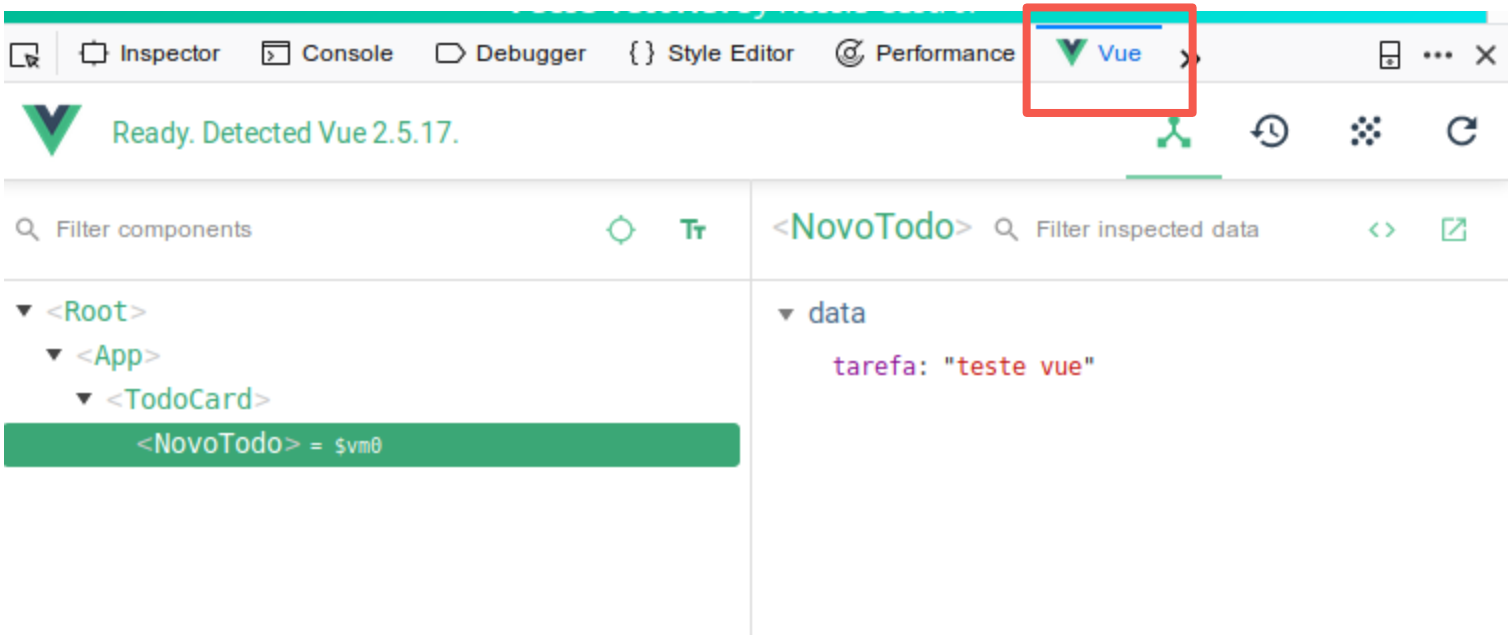
Vue Extensions

- For development install Vue.js devtools on Firefox or Chrome



Vue Extensions

- For development install Vue.js devtools on Firefox or Chrome



Vue communication between components

Vue communication between components

- Son -> Father
  - Son emit a event:

```
<p class="control">
  <a class="button is-danger is-small"  v-on:click="remover(index)">
    <span class="icon is-small">
      <i class="fa fa-trash"></i>
    </span>
  </a>
</p>
```

```
remover(index) {
  this.$emit('remover', index)
}
```

Vue communication between components

- Son -> Father
  - Father listener the event of son and call another function:

```
<todo-list v-bind:tarefas="tarefas" v-on:check="checkTarefa" v-on:remover="removerTarefa"></todo-list>
```

```
removerTarefa(index){
    this.tarefas.splice(index, 1)
}
```

Vue communication between components

- Father -> Son
  - Father pass value to son component:

```
<todo-list v-bind:tarefas="tarefas" v-on:check="checkTarefa" v-on:remover="removerTarefa"></todo-list>
```

  - Son receive the value of the Father component

```
props: ['tarefas'],
```

```
<div class="field is-grouped" v-for="(tarefa, index) in tarefas">
```

Vue.js Routes

Vue.js Routes

- Routers is the way you can "change" the page at single-page applications
- Create a project and install the vue-router library

```
vue init webpack-simple vue-routers
cd vue-routers
npm install
npm install vue-router --save
npm run dev
```

Vue.js Routes

- Create two components Bar and Foo
  - Create a file routes.js
  - At this file define a variable routes with path "/foo" to Foo component and "/bar" to Bar Component

```
import Bar from './components/Bar'
import Foo from './components/Foo'


const routes = [
    { path: '/foo', component: Foo },
    { path: '/bar', component: Bar }
]


export default routes
```

Vue.js Routes

- Now at the main.js:
  - Import the vue-router library,
  - Import the routes.js file
  - Declare Vue.use (VueRouter)
  - Create a na VueRuter variable and, define this variable at app component

Vue.js Routes

- Now at the main.js:

```
import Vue from 'vue'
import VueRouter from 'vue-router' // import the vue router
import routes from './routes'

import App from './App.vue'

/**
 * When used with a module system, you must explicitly install t
 */
Vue.use(VueRouter)

// 3. Create the router instance and pass the `routes` option
// You can pass in additional options here, but let's
// keep it simple for now.
const router = new VueRouter({
    routes // short for `routes: routes`
})

new Vue({
  el: '#app',
  router: router,
  render: h => h(App)
})
```

Vue.js Routes

- At app component, include <router-link> to /foo and /bar, and a <router-view> (where the components will be rendered):

```
<div id="app">

  <h1>Hello App!</h1>

  <p>
    <!-- use router-link component for navigation. -
    <!-- specify the link by passing the `to` prop.
    <!-- `<router-link>` will be rendered as an `<a>
    <router-link to="/foo">Go to Foo</router-link>
    <router-link to="/bar">Go to Bar</router-link>

  </p>

  <!-- route outlet -->
  <!-- component matched by the route will render he
  <router-view></router-view>

</div>
```

Vue.js Routes

- When click at the router-link /foo, the foo component is rendered, and click at the router-link /bar, the bar component is rendered:

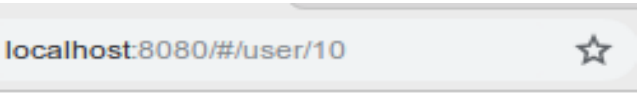
Vue.js Routes

- Children routes:

```
const routes = [
    { path: '/foo', component: Foo },
    { path: '/bar', component: Bar },
    { path: '/user/:id', component: User,
      children: [
        {
          // UserProfile will be rendered inside User's
          // when /user/:id/profile is matched
          path: 'profile',
          component: UserProfile
        },
        {
          // UserPosts will be rendered inside User's <
          // when /user/:id/posts is matched
          path: 'posts',
          component: UserPosts
        }
      ]
    }
]
```

Vue.js Routes

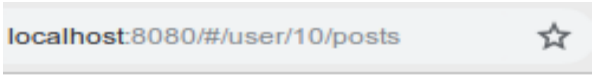
- Children routes:



Hello App!

Go to Foo Go to Bar

User 10

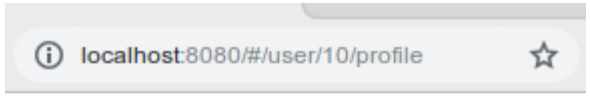


Hello App!

Go to Foo Go to Bar

User 10

List of Posts



Hello App!

Go to Foo Go to Bar

User 10

Profile

Vue.js connecting with the back-end

Vue.js connecting with the back-end

- Axios is a great http client library.
- It uses promises by default and runs on both the client and the server (which makes it appropriate for fetching data during server-side rendering).
- It's also quite easy to use with Vue. Because it uses promises, you can combine it with async/await to get an amazingly concise and easy-to-use API.

Vue.js connecting with the back-end

- Install Axios:

```
vue init webpack-simple vue-rest-api
cd vue-rest-api/
npm install
npm install axios --save
```

Vue.js connecting with the back-end

- Back-end:
  - spring boot application:

SPRING INITIALIZR bootstrap your application now

Generate a Gradle Project with Java and Spring Boot 2.0.5

Project Metadata

Artifact coordinates

Group

br.com.jadson.vue

Artifact

vue-backend

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Web JPA H2

Generate Project alt + ⏎

Vue.js connecting with the back-end

- Back-end:
  - spring boot rest controller:

```
@RestController
public class UserResource {

    @Autowired
    UserRespository userRepository;


    @GetMapping("/user/{id}")
    public User getUser(@PathVariable(value="id") Long id) {
        return userRepository.findById(id).orElse( new User());
    }


    @GetMapping("/users")
    public List<User> list() {
        return userRepository.findAll();
    }


    @PostMapping(path="/user", consumes = MediaType.APPLICATION_JSON_VALUE)
    public User create(@RequestBody @Valid User user) {
        return userRepository.save(user);
    }

}
```

Vue.js connecting with the back-end

- Back-end:
  - Allow CORS origing for spring rest controllers.
  - Front-end and back-end will run at different address, so we need to enable this configuration or our request will be block

```
@Configuration
public class MyConfiguration {

    /**
     * Enabling CORS for the whole application
     * @return
     */
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**");
            }
        };
    }
```

Vue.js connecting with the back-end

- Front-end:
  - Create a file http-common.js and declare an AXIOS constant and import it on App.vue component

```
import axios from 'axios';

export const HTTP = axios.create({
  baseURL: 'http://localhost:8180/',
  headers: {
    'Authorization': 'Bearer {token}',
    'Content-Type': 'application/json'
  }
})
```

```
import {HTTP} from './http-common.js'
```

Vue.js connecting with the back-end

- Front-end:
  - Declare data to hold the information, a **user object** and an **array of users**.

```
export default {
  name: 'app',
  data () {
    return {
      user: {
          id: 0,
          name: "",
          email: "",
          createAt: ""
        },
      users: [],
      errors: []
    }
  },
```

Vue.js connecting with the back-end

- Front-end:
  - Configure fields to this data and action to call vue methods that will call the back-end REST service

```
<div>
    <h1>Create a new User</h1>
    <input type="text" v-model="user.name" placeholder="Name" />
    <input type="text" v-model="user.email" placeholder="Email" />
    <button v-on:click="createUser2()">Send</button>
    <br />
    <br />
</div>


<button v-on:click="getUsers()">Get All Users</button>

<div>
    <h1>Get User:</h1>
    <input type="text" v-model="user.id" placeholder="Name" />
    <button v-on:click="getUser2()">Send</button>
    <br />
    <br />
</div>
```

Vue.js connecting with the back-end

- Front-end:
  - Create vue methods, and use the axios instance HTTP to call get and post back-end services.

```
getUser2() {
  HTTP.get("/user/"+this.user.id)
    .then(
      result => { console.log(result.data); this.users = []; this.users.push(result.data)}
    )
  .catch(
    e => { this.errors.push(e) }
  )
},
```

```
createUser2() {
    HTTP.post("http://localhost:8180/user", this.user)
    .then(
      result => { console.log(result.data);}
    )
  .catch(
    e => { this.errors.push(e) }
  )
}
```

Vuex

Vuex

- If your Vue.js application **grows bigger** and consists of multiple components you might run into the problem of how to share data across the those components and make sure that components which are using the same data are always updated if data is changing
- Vuex is a library to use as **a centralized state management** in your application.

Vuex

- Vuex Store concept:
  - **State Tree**: An object containing the data
  - **Getters**: Used to access data from the state tree of the store
  - **Mutations**: Handler functions that perform modifications of data in the state tree
  - **Actions**: Functions that commit mutations. The main difference to Mutations is that Actions can contain asynchronous operations

Vuex

- Installation
  - Include via CDN (Content Delivery Network):

    <script src="https://unpkg.com/vue@2.5.17/dist/vue.js"></script>

    <script src="https://unpkg.com/vuex@3.0.1/dist/vuex.js"></script>
  - Or via npm:

    vue init webpack-simple vue-vuex

    cd vue-vuex/

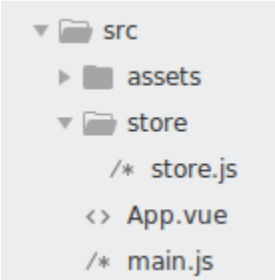
    npm install

    npm install vuex - - save

    npm run dev

Vuex

- Within the src folder create a new subfolder named store. Within that folder create a new **file store.js** and insert the following code:




```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

const state = {
  count: 0
}

const mutations = {
  increment (state) {
    state.count++
  },
  decrement (state) {
    state.count--
  }
}

export default new Vuex.Store({
    state,
    mutations
})
```

Vuex

- First you need to import store. Next, add store to the configuration object which is passed to **the Vue constructor.** By providing the store option to the root instance, the **store will be injected into all child components** of the root and will be available on them as **this.$store**:

```
import Vue from 'vue'
import App from './App.vue'

import store from './store/store'

new Vue({
  el: '#app',
  store,
  render: h => h(App)
})
```

Vuex

- Now we can implement a simple counter:

```
<template>
  <div id="app">
  <p> {{ $store.state.count }} </p>
  <p>
    <button v-on:click="increment">+</button>
    <button v-on:click="decrement">-</button>
  </p>
</div>
</template>

<script>
export default {
  name: 'app',
  methods: {
    increment () {
        this.$store.commit('increment')
    },
    decrement () {
        this.$store.commit('decrement')
    }
  }
}
```

10

+ -

Vuex

- Keep vuex data between request
  - npm install vuex-persistedstate
  - npm install vue-cookies

```
import PersistedState from 'vuex-persistedstate'
import VueCookies from 'vue-cookies'


Vue.use(Vuex)
Vue.use(VueCookies)


export default new Vuex.Store({
    modules: {
        a: moduleA
    },
    plugins: [
        PersistedState({
            getState: (key) => window.$cookies.get(key),
            setState: (key, state) => window.$cookies.set(key,state)
        })
    ]
})
```

Vue Pre-rendering

Vue Pre-rendering

- One of the downsides to Javascript-based apps is that the browser receives an **essentially empty page from the server**. The DOM cannot be built until the Javascript has been downloaded and run.

- It have an impact on SEO (Search Engine Optimization) if crawlers can't see content of the page quickly

Vue Pre-rendering

- **Server-side rendering** (SSR) overcomes this issue by rendering the app on the server so that the client receives the complete DOM content when the page is loaded, before Javascript is even run.
  - Your app will need to be executable on the server,
  - Your app will run on each request to the server, adding aditional load and slowing responses
  - You can only do SSR with Node.js

Vue Pre-rendering

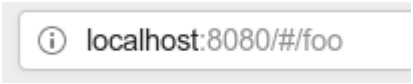
- There's another way to tackle the empty page problem: **pre-rendering**.
  - With this approach you run your app before deploying it, capture the page output and replace your HTML files with this captured output
  - It's pretty much the same concept as SSR except it's done pre-deployment in your development environment, not a live server

Vue Pre-rendering

- Pre-rendering pros
  - No additional server load,
  - A simpler production setup and simpler app code
  - **Doesn't require a Node.js production server**
- Pre-rendering cons
  - **Doesn't work for pages that display changing data** e.g. tables
  - Doesn't work for pages that have user-specific content e.g. an account page with a user's personal details
  - You'll need to pre-render every route in the app indiviually.

Vue Pre-rendering

- Let's take our vue-router project with 2 routers /foo and /bar



Hello App!

Go to Foo Go to Bar

Foo

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>vue-routers</title>
  </head>
  <body>
    <div id="app"></div>
    <script src="/dist/build.js"></script>
  </body>
</html>
```

Vue Pre-rendering

- Step 1: There are three additional modules we'll need to install

```
npm install   http-server   html-webpack-plugin
prerender-spa-plugin    --save
```

Vue Pre-rendering

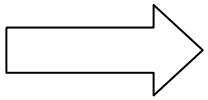
- Step 2: Include Webpack and PrerenderSpa plugins
  - Open the **webpack.config.js file** and include http-webpack-plugin and prerender-spa-plugin

```
var path = require('path')
var webpack = require('webpack')

/* *** to configurate vue with prerender *** */

var HtmlWebpackPlugin = require('html-webpack-plugin');
var PrerenderSpaPlugin = require('prerender-spa-plugin');
```

Vue Pre-rendering

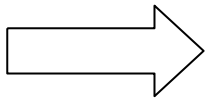
- Step 3: Change the build.js location
  - Now we change our Webpack "publicPath" in same **webpack.config.js** file since the index.html will now be in the same folder as the other static assets
  - And we'll also need to change location of the build.js file in our **index.html** file due to the changed path

```
module.exports = {
  entry: './src/main.js',
  output: {
    path: path.resolve(__dirname, './dist'),
    publicPath: '/dist/',
    filename: 'build.js'
  },
```



```
module.exports = {
  entry: './src/main.js',
  output: {
    path: path.resolve(__dirname, './dist'),
    publicPath: '/',
    filename: 'build.js'
  },
```

```
<body>
  <div id="app"></div>
  <script src="/dist/build.js"></script>
</body>
```



```
<body>
  <div id="app"></div>
  <script src="/build.js"></script>
</body>
```

Vue Pre-rendering

- Step 4: Include index.html and add pre-render
  - The webpack-simple template doesn't include the index.html file in the Webpack build output. However when we pre-render the app we'll need to overwrite our index.html, so let's add it to the output so as not to destroy the original
  - Now we need to add prerender-spa-plugin to our webpack config. Make sure it comes after html-webpack-plugin
    - New add the index.html and our routes
    - The first argument to PrerenderSpaPlugin is the location of our index.html file, the second is a list of routes in the app.
    - **For each route we add, we'll get a different output file!**

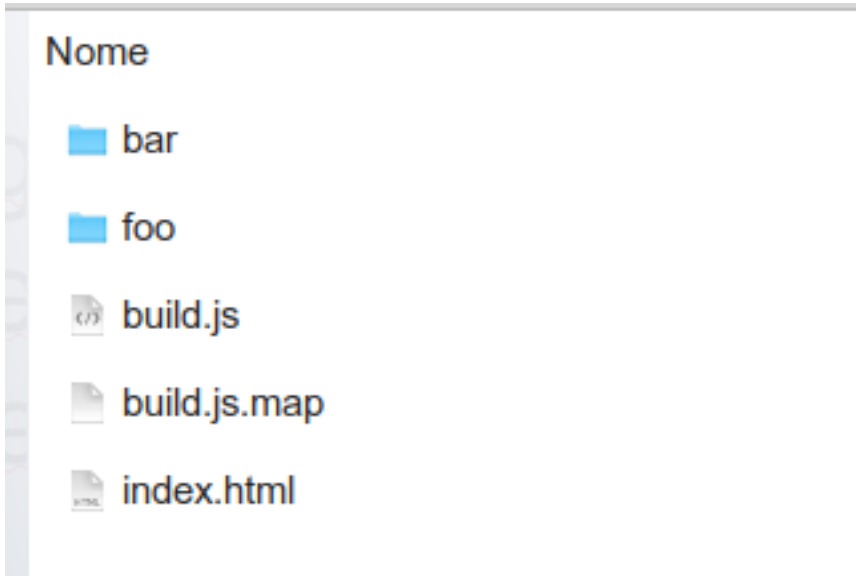
Vue Pre-rendering

- Step 4: Include index.html and add pre-render

```
if (process.env.NODE_ENV === 'production') {
  module.exports.devtool = '#source-map'
  // http://vue-loader.vuejs.org/en/workflow/production.html
  module.exports.plugins = (module.exports.plugins || []).concat([
    new webpack.DefinePlugin({
      'process.env': {
        NODE_ENV: '"production"'
      }
    }),
    new webpack.optimize.UglifyJsPlugin({
      sourceMap: true,
      compress: {
        warnings: false
      }
    }),
    new webpack.LoaderOptionsPlugin({
      minimize: true
    }),
    /* *** to configurate vue with prerender *** */
    new HtmlWebpackPlugin({
      template: './index.html',
      inject: false
    }),
    new PrerenderSpaPlugin(
      path.join(__dirname, './dist'),
      [ '/', '/foo', '/bar' ]
    )
    /* ************************************** */
  ])
```

Vue Pre-rendering

- Step 5: Building
  - Now make the build for production
    - npm run build
  - It will generated one folder for each route with the static html code



Vue Pre-rendering

- Step 6: Running
  - You can run the application inside dist folder using the http-server

    ./node_modules/.bin/http-server ./dist/
  - It generated the static content of page, but not the dynamic information.
  - The "Foo" word is not present

Awesome Vue

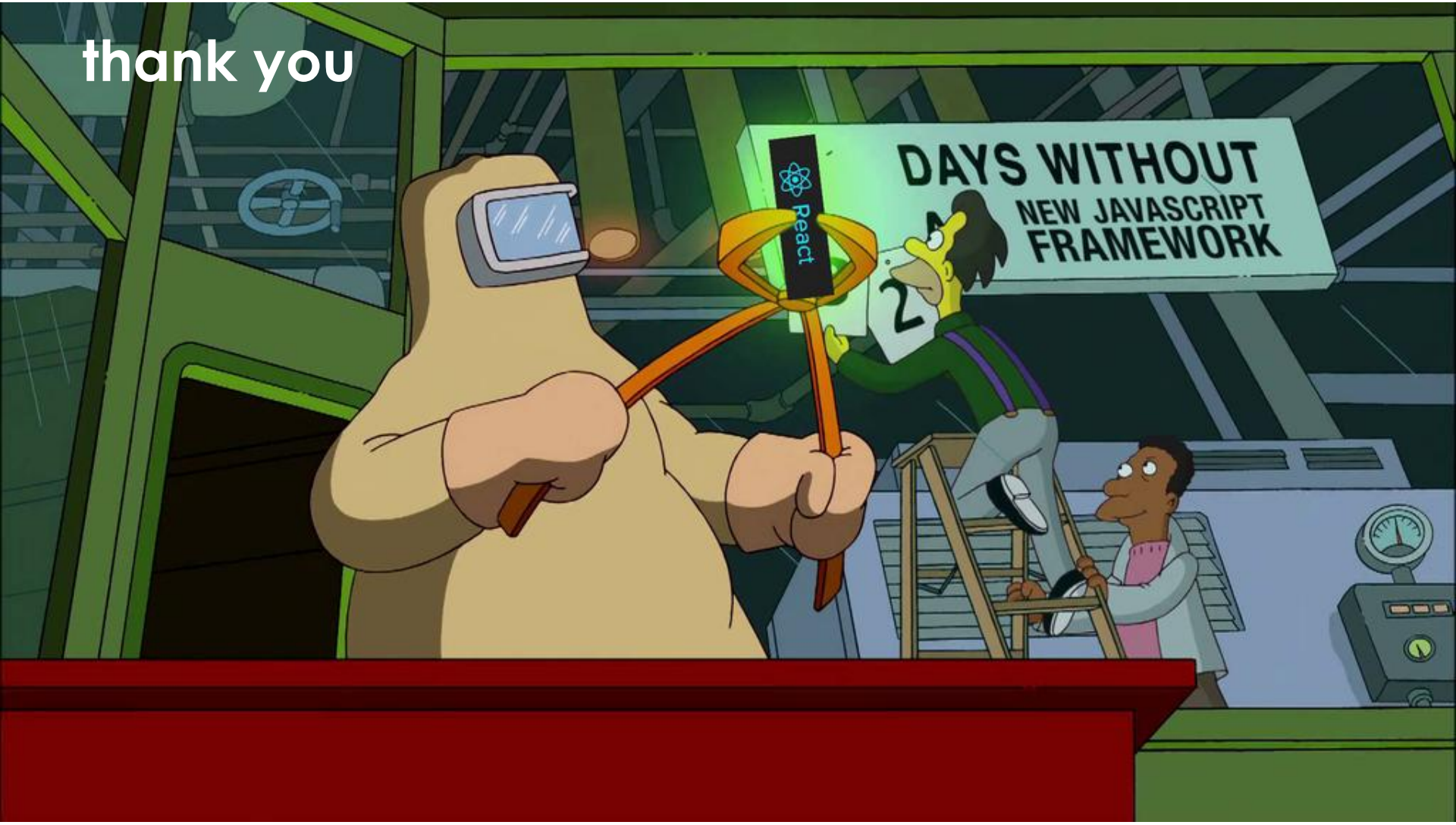
https://github.com/vuejs/awesome-vue

- **Source Code**

  - https://github.com/jadsonjs/vuejs

- **References**
  - https://vuejs.org/v2/guide/
  - https://www.devmedia.com.br/vue-js-tutorial/38042
  - https://www.shopify.com/partners/blog/vuejs-tutorial
  - https://github.com/vuejs-templates/webpack
  - http://www.vuejs-brasil.com.br/crie-rapidamente-um-projeto-vue-com-vue-cli-e-browserify/
  - https://medium.com/@thiagorossener/como-fazer-um-hello-world-com-vue-js-2-62ea708c1399

- **References**
  - http://vegibit.com/vue-js-directives/
  - https://alligator.io/vuejs/rest-api-axios/
  - https://www.thepolyglotdeveloper.com/2017/10/consume-api-data-http-vuejs-web-application/
  - https://medium.com/codingthesmartway-com-blog/vue-js-2-state-management-with-vuex-introduction-db26cb495113
  - https://vuejsdevelopers.com/2017/04/01/vue-js-prerendering-node-laravel/



source: https://blog.hellojs.org/differences-between-angularjs-vuejs-and-reactjs-5d80203dfd16