



VueJS and Laravel

Part 1 : Introduction to VueJS



django



10 BEST WEB DEVELOPMENT FRAMEWORKS

Framework & Libraries

Libraries & Framework	Language
Django	Framework using Python
Spring	Framework using Java
Laravel	Framework using PHP
Rails	Framework using Ruby
React JS	Frontend Library using JS (SPA Framework)
Angular	Framework using JS (SPA Framework)
Express	Backend Framework using JS
Vue JS	Framework using JS (SPA Framework)
Jquery	Frontend Library using JS

Why Vue

- Low learning curve
- Widely use or **rising technology**
- Nearly as powerful as other framework (ReactJS, Angular)
- Very modular / flexible
- New, but stable technology

Getting started with Vue JS

Install Nodejs and npm

<https://nodejs.org/en/>

To check if nodeJS is installed use - `npm --version`

Install Vue

`vue --version`

`npm install --global @vue/cli` (PC)

`sudo npm install --global @vue/cli` (MAC)

To create and Start the project

```
Vue create hello-vue
```

```
Select default (Vue?)
```

If everything is ok this page should come out:

← → ↻ localhost:8080 ☆ ⓘ 🗨️ 🌐 📄 🛠️ 🌍 ⋮



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

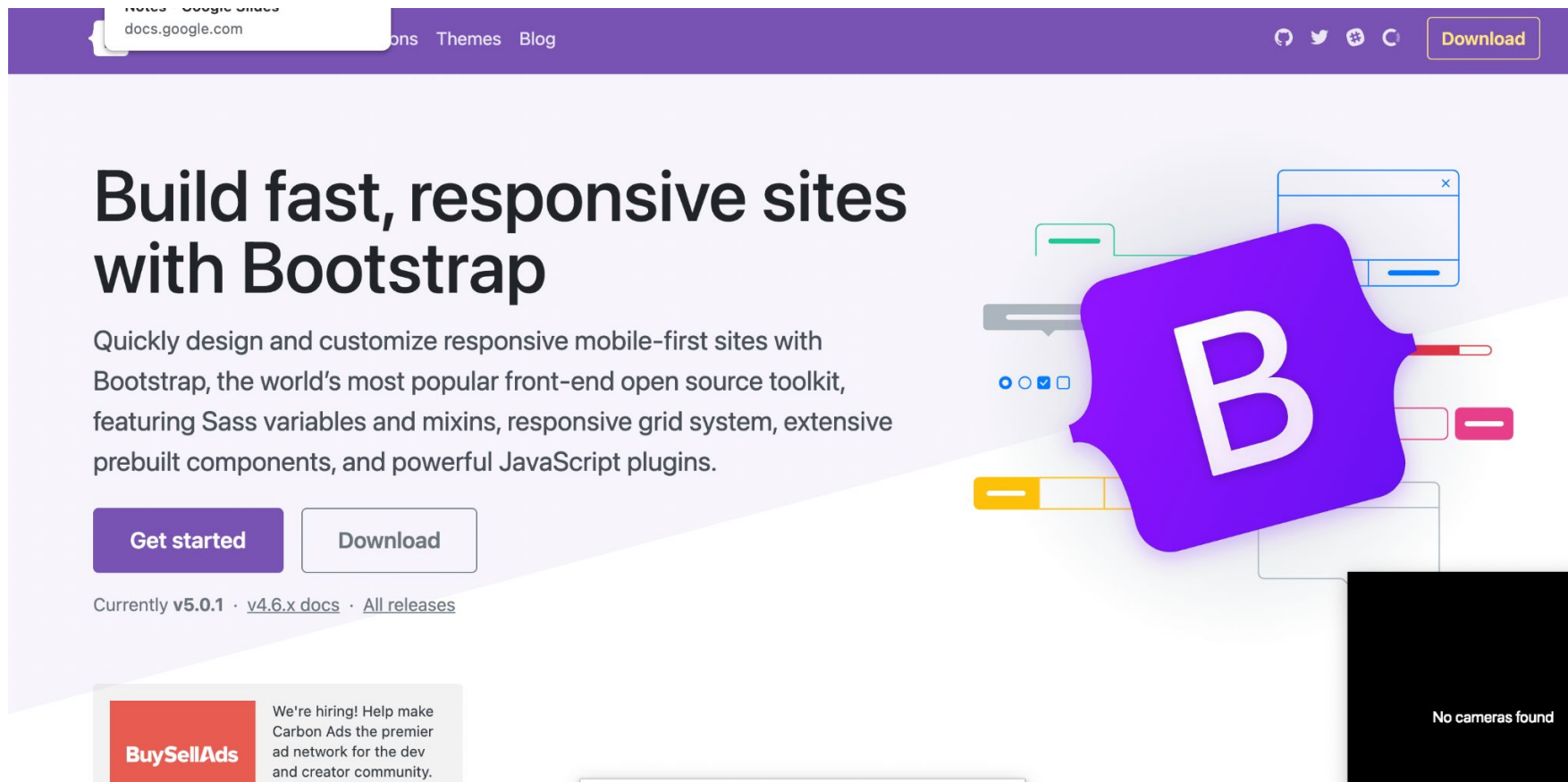
Ecosystem

No cameras found

Install vue.js plugin on Sublime

- 1) Ctrl + shift + p / cmd + shift + p -> Install Package Control
- 2) Ctrl + shift + p / cmd + shift + p -> Install package
- 3) Wait for 5-10s another window will pop up, then you look for **vue complete package**
- 4) Go down right and look for Vue Component (from plain text)

Retrieve Bootstrap code from : <https://getbootstrap.com>



The image shows a screenshot of the Bootstrap website homepage. The page has a purple header with navigation links for 'Home', 'Features', 'Components', 'Themes', and 'Blog'. A 'Download' button is visible in the top right corner. The main content area features the heading 'Build fast, responsive sites with Bootstrap' and a subheading 'Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.' Below this are two buttons: 'Get started' and 'Download'. At the bottom left, there is an advertisement for 'BuySellAds' with the text 'We're hiring! Help make Carbon Ads the premier ad network for the dev and creator community.' At the bottom right, there is a black box with the text 'No cameras found'.

docs.google.com

Home Features Components Themes Blog

Download

Build fast, responsive sites with Bootstrap

Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end open source toolkit, featuring Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful JavaScript plugins.

[Get started](#) [Download](#)

Currently **v5.0.1** · [v4.6.x docs](#) · [All releases](#)

BuySellAds We're hiring! Help make Carbon Ads the premier ad network for the dev and creator community.

No cameras found

Retrieve bootstrap code and add inside index.html

Getting started

Introduction

Download

Contents

Browsers & devices

JavaScript

Build tools

Webpack

Parcel

Accessibility

RFS

RTL

Customize

Layout

Content

Forms

Components

Helpers

Introduction

[View on GitHub](#)

Get started with Bootstrap, the world's most popular framework for building responsive, mobile-first sites, with jsDelivr and a template starter page.

BuySellAds

We're hiring! Help make Carbon Ads the premier ad network for the dev and creator community.
ads via Carbon

Quick start

Looking to quickly add Bootstrap to your project? Use jsDelivr, a free open source CDN. Using a package manager or need to download the source files? [Head to the downloads page.](#)

CSS

Copy-paste the stylesheet `<link>` into your `<head>` before all other stylesheets to load our CSS.

On th

Quick

CSS

JS

B

S

M

C

Starter

Import

HTM

Res

Box

Reb

Comm

Add it in index.html

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.1/dist/css/bootstrap.min.css" rel="stylesheet">
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
```

Starter code

```
1 <template>
2   <div id="app">
3
4   </div>
5 </template>
6
7 <script>
8
9   export default {
10    name: 'App',|
11
12  }
13 </script>
14
15 <style>
16 #app {
17   font-family: Avenir, Helvetica, Arial, sans-serif;
18   -webkit-font-smoothing: antialiased;
19   -moz-osx-font-smoothing: grayscale;
20   text-align: center;
21   color: #2c3e50;
22   margin-top: 60px;
23 }
24 </style>
```

To verify if Bootstrap is loaded

```
<template>  
  <div class="container">  
    <h1>Hello World</h1>  
  </div>  
</template>
```

```
<script>  
  export default {  
    name: 'App'  
  }  
</script>
```

```
<style>
```

```
</style>
```

Hello Vue! (template)

```
<template>
```

```
<div id="app">
```

```
  {{msg}}
```

```
</div>
```

```
</template>
```

Hello Vue! (script)

```
<script>
```

```
  export default {
```

```
    name: 'App',
```

```
    data() {
```

```
      return {
```

```
        msg: "Hello World",
```

```
      }
```

```
    }
```

```
  }
```

```
</script>
```

Creating a vue instance

- 1) Start creating a vue instance as follows:

```
var app = new Vue({  
  
  // options  
  
})
```

1- Create variables inside script

```
<script>
  export default {
    name: 'App',
    data() {
      return {
        firstname : "Wan Muzaffar",
        lastname  : "Wab Hashim",
        htmlcontent : "<div><h1>Hello vue!</h1></div>"
      }
    }
  }
</script>
```



```
<script>
export default {
  name: 'App',
  data(){
    return {
      // 5 main data types = String , number, boolean
      // array and object
      name:"Muzaffar",
      location:"Bangi",
      age:30,
      hasBreakfast:false,
      scores:[70,90,70,60,65],
      companyInfo:{"name":"Anak2U Sdn Bhd",
        "smmNum":"xxxxxxxx",
        "location":"Bangi"}
    }
  }
}
</script>
```

2 - Rendering with `{{}}` in html part

```
<div id = "vue_det">
```

```
  <h1>Firstname : {{firstname}}</h1>
```

```
  <h1>Lastname : {{lastname}}</h1>
```

```
  <div>{{htmlcontent}}</div>
```

```
</div>
```

Event Handling in Vuejs

```
<div id="example-1">
```

```
<button v-on:click="counter += 1">Add 1</button>
```

```
<p>The button above has been clicked {{ counter }} times.</p>
```

```
</div>
```

```
var example1 = new Vue({
```

```
  el: '#example-1',
```

```
  data: {
```

```
    counter: 0
```

```
  }
```

```
})
```

Event Handling with methods (1)

```
<div id="example-2">
```

```
  <!-- `greet` is the name of a method defined below -->
```

```
  <button v-on:click="greet">Greet</button>
```

```
</div>
```

Event handling with methods (2)

```
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // define methods under the `methods` object
  methods: {
    greet: 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)
      }
    }
  }
})
```

```
},  
methods:{  
  sayHello: function(){  
    alert("Hello World");  
  }  
}  
}
```

Call back the function

```
<template>
  <div class="container">
    <h1>Hello World</h1>
    <p>My name is {{name}} and I stay in {{location}} </p>
    <p>I am {{age}} years old</p>
    <button v-on:click="sayHello">Say Hello</button>
  </div>
</template>

<script>
export default {
```

Conditional rendering - v-if

The directive `v-if` is used to conditionally render a block. The block will only be rendered if the directive's expression returns a truthy value. (Vuejs, Angular - ng)

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

You may also add an else statement for the block:

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

```
<h1 v-else>Oh no 😞</h1>
```

Ternary operator (React.Js, Flutter , Laravel)

```
<condition> ? <if true i will do this> : <if else I will do this>
```


Binding variables to attributes using v-bind

We use v-bind when we want to link the *attribute* of an *element* to a *property/variable* of class (data)

Mustaches cannot be used inside HTML attributes. Instead, use a **v-bind** directive.

```
<div v-bind:id="dynamicId"></div>
```

HTML

```
<button v-bind:disabled="isButtonDisabled">Button</button>
```

HTML

```
<a v-bind:href="url"> ... </a>
```

HTML

HTML recap (definition)

Element = `<p></p>`, `<a>` , `<div></div>`

Attribute = everything after space = ``, `<div id=""/>`

Entity: The symbols : `™` , `©` , `>` , `<`

List rendering - v-for (1)

We can use the `v-for` directive to render a list of items based on an array.

(singular) in (plura)

```
<ul id="example-1">
```

```
  <li v-for="item in items" :key="item.message">
```

```
    {{ item.message }}
```

```
  </li>
```

```
</ul>
```

V-for with index - `<li v-for="(score, index) in scores" :key="index">{{score}}`

List rendering - v-for (2)

```
var example1 = new Vue({  
  el: '#example-1',  
  data: {  
    items: [  
      { message: 'Foo' },  
      { message: 'Bar' }  
    ]  
  }  
})
```

Data binding with v-model

You can use the `v-model` directive to create two-way data bindings on form input, textarea, and select elements. It automatically picks the correct way to update the element based on the input type.

Example:

```
<input v-model="message" placeholder="edit me">
<p>Message is: {{ message }}</p>
```

```
<span>Multiline message is:</span>
```

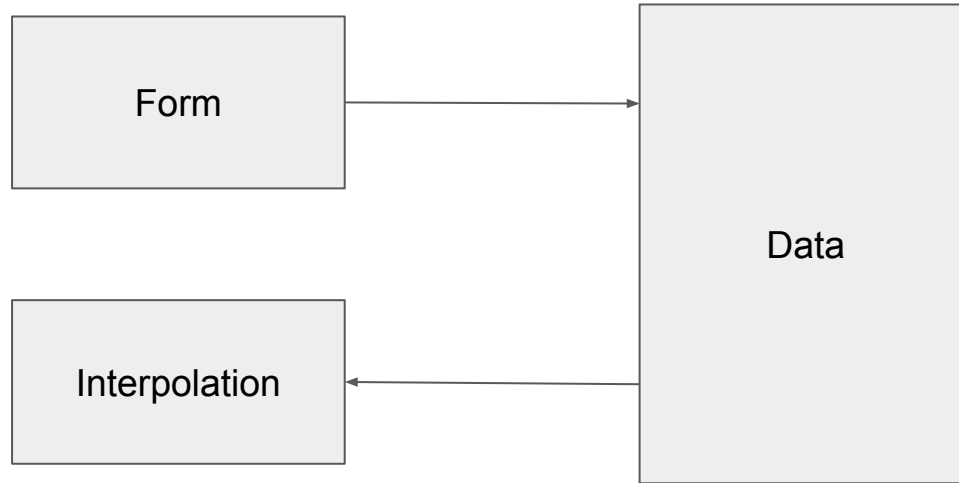
```
<p style="white-space: pre-line;">{{ message }}</p>
```

```
<br>
```

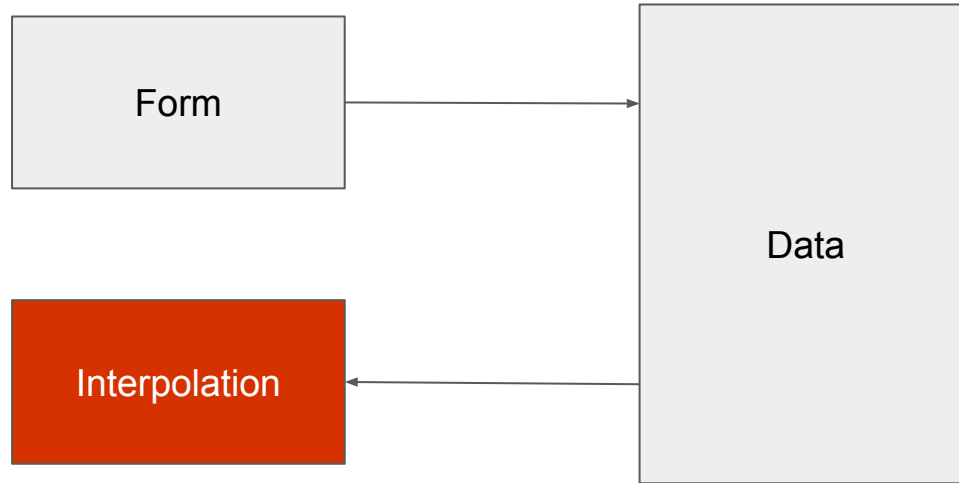
```
<textarea v-model="message" placeholder="add multiple lines"></textarea>
```

More example : <https://vuejs.org/v2/guide/forms.html>

Two way data binding (VueJS, Angular)



One way data binding (React.js, Flutter)



*If you want to update the page, normally you will need to call the attribute `onChangeText` (Flutter), `onValueChange`...

Class binding

We can pass an object to `v-bind:class` to dynamically toggle classes:

```
<div
  class="static"
  v-bind:class="{ active: isActive, 'text-danger': hasError }"
></div>
```

```
data: {
  isActive: true,
  hasError: false
}
```


Recap Bootstrap

Form:

```
<input type="text" placeholder="Enter Exercise name"  
      class="form form-control mb-3" />
```

Button:

```
<button class="btn btn-primary">Add</button>    primary, secondary, danger, success, info  
...
```

Table:

```
<table class="table table-bordered table-striped table-hover"></table>
```

Basic Vue JS

1. Interpolation `{{}}` - to bring out the data (string, number, object)
2. `V-on:click` -> button click -> to call the methods
3. `V-model` -> link the input to the data
4. `v-for` => for loop , bring out data from (array) [.. replacing `createElement...`], need to specify the key and the key needs to be unique
5. `v-if` -> conditional rendering -> bringing out data based on certain condition (you play with boolean data)
6. `v-bind` - to attach variable to attribute (src, href, disabled, hidden...)
7. `mounted` -> This is the hook that's going to be called when the page is loaded...

Local Storage

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

To save the data we will use:

```
localStorage.setItem("todos", JSON.stringify(this.todos));
```

We need to save the data in either string, number or boolean, JSON.stringify will change the value from array/object to string

https://www.w3schools.com/html/html5_webstorage.asp

https://www.w3schools.com/js/js_json_stringify.asp

Local Storage (2)

To retrieve the data we will use:

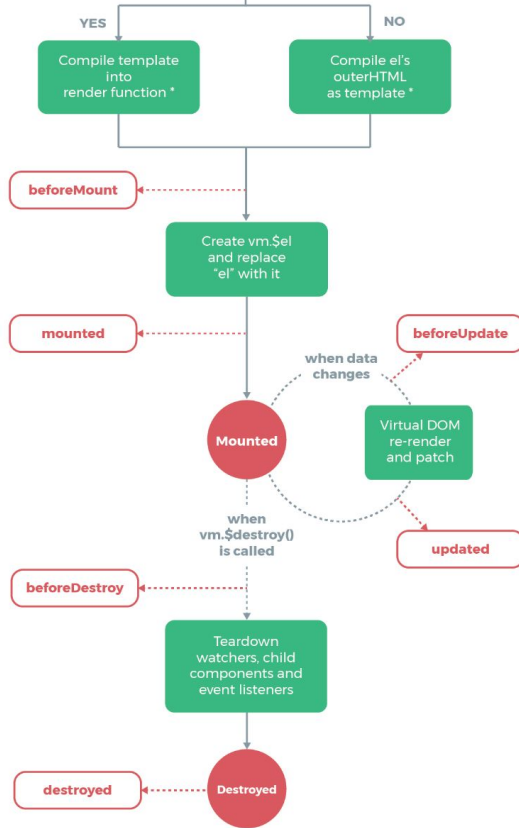
```
localStorage.getItem("todos") ;
```

JSON.parse will change the value from String to array and object

https://www.w3schools.com/html/html5_webstorage.asp

https://www.w3schools.com/js/js_json_parse.asp

Vue Lifec



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

<https://vuejs.org/v2/guide/instance.html>

Most of the SPA (VueJS, Angular, ReactJS) will have hook...

Originally it was from mobile

Vue JS hook

Hook	Explanation
<code>beforeCreate</code>	Runs at the very initialization of your component. <code>data</code> has not been made reactive, and <code>events</code> have not been set up yet
created	You are able to access reactive <code>data</code> and <code>events</code> that are active with the <code>created</code> hook.
<code>beforeMount</code>	Called right before the mounting begins: the <code>render</code> function is about to be called for the first time.
mounted	Called after the instance has been mounted, note that <code>mounted</code> does not guarantee that all child components have also been mounted.
<code>beforeUpdate</code>	Called when data changes, before the DOM is patched. This is a good place to access the existing DOM before an update
updated	Called after a data change causes the virtual DOM to be re-rendered and patched.
<code>beforeDestroy</code>	<code>beforeDestroy</code> is fired right before teardown. Your component will still be fully present and functional.
destroyed	By the time you reach the <code>destroyed</code> hook, there's practically nothing left on your component. Everything that was attached to it has been destroyed.

Segrating pages into multiple components (1)

```
1 <template>
2   <div class="hello">
3     <h1>{{ msg }}</h1>
4
5   </div>
6 </template>
7
8 <script>
9   export default {
10    name: 'HelloWorld',
11    data() {
12      return {
13        msg: "This is another page!"
14      }
15    }
16  }
17 </script>
18
```

Inside new components, export the items (take attention on the exported name)

The Moose Academy

Introduction to VueJS component

What is a components

Components are reusable Vue instances with a name: in this case, `<button-counter>`. We can use this component as a custom element inside a root Vue instance created with `new Vue`:

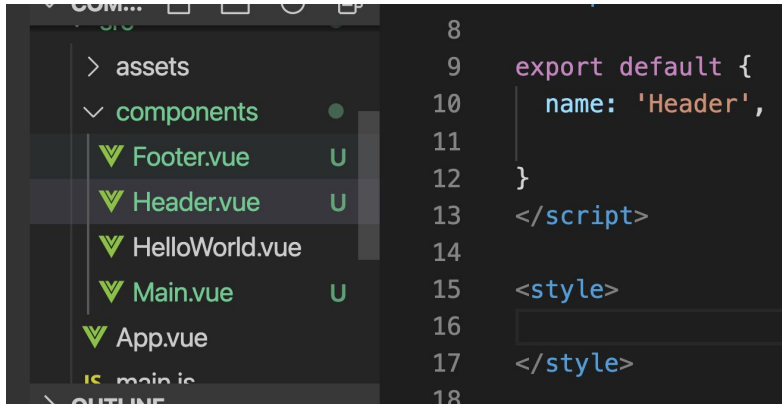
```
// Define a new component called
button-counter
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button
v-on:click="count++">You clicked me {{
count }} times.</button>'
})
```

```
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

Centralized all in components folder

Create three files which are

- Header.vue
- Main.vue
- Footer.vue



```
8
9  export default {
10     name: 'Header',
11   }
12 }
13 </script>
14
15 <style>
16
17 </style>
18
```

Based of a component.. Change the component name to the filename

```
<template>
```

```
  <div>
```

```
    </div>
```

```
</template>
```

```
<script>
```

```
export default {
```

```
  name: 'Header',
```

```
}
```

```
</script>
```

Header code (Bootstrap + HTML5 revision)

```
<template>
```

```
  <header class="bg-primary text-center p-5">
```

```
    <h1>My Vue app</h1>
```

```
  </header>
```

```
</template>
```

Footer code (Bootstrap + HTML5 revision)

```
<template>  
  <footer class="text-center bg-dark text-light">  
    &copy; Copyright 2021  
  </footer>  
</template>
```

Simple Main page

```
<template>
```

```
  <main>
```

```
    <h2>Hello World</h2>
```

```
  </main>
```

```
</template>
```

In App.vue, we will import and call the components:

```
<script>
import Header from './components/Header'
import Footer from './components/Footer'
import Main from './components/Main'

export default {
  name: 'App',
  components: {
    Header,
    Main,
    Footer
  }
}
</script>
```

App.vue the template part

```
<template>
```

```
  <div>
```

```
    <Header/>
```

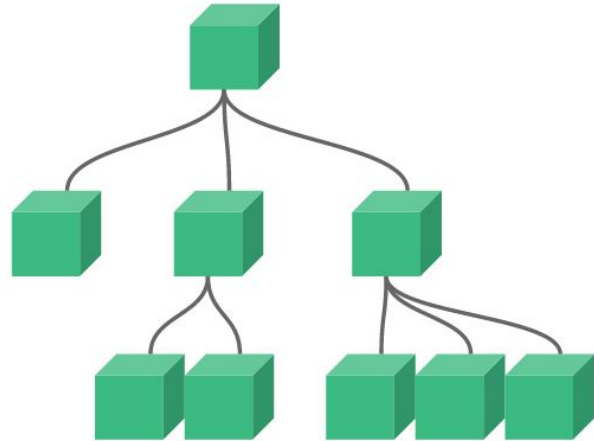
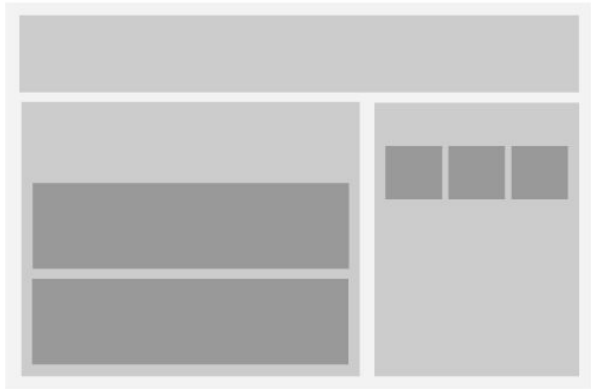
```
    <Main/>
```

```
    <Footer/>
```

```
  </div>
```

```
</template>
```


Organizing Components



Create a component (Blog.Vue)

```
<template>
<article>
<h2>This is the title</h2>
<h6>This is the subtitle</h6>
<p>THIS is example of a blog.....</p>
</article>
</template>
```

```
<script>

export default {
name: 'Blog',
}
</script>
```

```
<style>
```

Call Blog inside the main components...

```
<script>
import Blog from './Blog'
export default {
  name: 'Main',
  components: {
    Blog
  }
}
</script>
```

```
<script>
import Blog from './Blog'
export default {
  name: 'Main',
  components: {
    Blog
  }
}
</script>
```

Add props in Blog.vue

```
<script>
export default {
  name: 'Blog',
  props: ['title', 'subtitle', 'content']
}
</script>
```

You will bring out this props on the HTML

```
<script>  
export default {  
  name: 'Blog',  
  props: ['title', 'subtitle', 'content']  
}  
</script>
```

Pass data from Main.vue to Blog.vue through props

```
<Blog title="My first blog"  
  subtitle="Happy to blog"  
  content="Wilkomen !!"/>
```

We create an array of blogs ...

```
<script>
import Blog from './Blog'
export default {
  name: 'Main',
  components: {
    Blog
  },
  data(){
    return {
      blogs:[
        {
          "title":"My first blog",
          "subtitle":"Happy to blog",
          "content":"Wilkommen !!"
        },
        ..
      ]
    }
  }
}
</script>
```

And we pass the data from Array to Blog using v-for

```
<Blog v-for="blog in blogs"  
  :key="blog.title"  
  v-bind:title="blog.title"  
  v-bind:subtitle="blog.subtitle"  
  v-bind:content="blog.content"/>
```


For improvement, we will pass the object instead of individual property

```
<Blog v-for="blog in blogs"  
  :key="blog.title"  
  v-bind:blog="blog"  
>
```

The change on Blog.vue

```
<template>
<div class="card p-5 bg-light m-3">
<article>
<h2>{{blog.title}}</h2>
<h6>{{blog.subtitle}}</h6>
<p>{{blog.content}}</p>
</article>
</div>
</template>
```

```
<script>

export default {
  name: 'Blog',
  props: ['blog']
}
</script>
```

Pass up data (In Blog.vue)

```
export default {
  name: 'Blog',
  props: ['blog'],
  methods: {
    buttonPressed() {
      this.$emit('deleted')
    }
  }
}
</script>
```

In the component, pass it as a prop

```
<Blog v-for="blog in blogs"  
  :key="blog.title"  
  v-bind:blog="blog"  
  v-on:deleted="onDeleteClicked"  
>
```

In Main.vue, create the method onDeleteClicked

```
export default {
  name: 'Main',
  components: {
    Blog
  },
  methods: {
    onDeleteClicked() {
      alert('clicked from parent!')
    }
  },
  data() {

  ..

}
```

Pass the parameter in child

```
export default {
  name: 'Blog',
  props: ['blog'],
  methods: {
    buttonPressed() {
      this.$emit('deleted', this.blog.title)
    }
  }
}
</script>
```

Retrieve it from parent via the parameter of method

```
export default {
  name: 'Main',
  components: {
    Blog
  },
  methods: {
    onDeleteClicked(val) {
      alert('clicked from parent! data passed is '+val)
    }
  },
  data() {
    Return ..
  }
}
```

Update with delete code

```
export default {
  name: 'Main',
  components: {
    Blog
  },
  methods: {
    onDeleteClicked(val) {
      this.blogs = this.blogs.filter(item=>{
        return item.title !== val
      })
    }
  },
}
```


React JS / Vue / Angular comparison

Angular	ReactJS
interpolation	interpolation
onclick	v-onclick
v-model	(one way data binding)
v-for	(use map)
v-if	(use ternary operator)
v-bind	???
Mounted (ngOnInit)	useEffect
props..	Props [more closer to VueJS

Components reusable

Components can be reused as many times as you want:

```
<div id="components-demo">  
  <button-counter></button-counter>  
  <button-counter></button-counter>  
  <button-counter></button-counter>  
</div>
```

Each one maintains its own, separate `count`. That's because each time you use a component, a new instance of it is created.

Passing data to Child Components with props

Component won't be useful unless you can pass data to it, such as the title and content of the specific post we want to display. That's where props come in.

Props are custom attributes you can register on a component. When a value is passed to a prop attribute, it becomes a property on that component instance. To pass a title to our blog post component, we can include it in the list of props this component accepts, using a `props` option:

Once a prop is registered, you can pass data to it as a custom attribute

```
Vue.component('blog-post', {  
  props: ['title'],  
  template: '<h3>{{ title }}</h3>'  
})
```

```
<blog-post title="My journey with  
Vue"></blog-post>  
<blog-post title="Blogging with  
Vue"></blog-post>  
<blog-post title="Why Vue is so  
fun"></blog-post>
```

Another example

```
new Vue({
  el: '#blog-post-demo',
  data: {
    posts: [
      { id: 1, title: 'My journey with Vue' },
      { id: 2, title: 'Blogging with Vue' },
      { id: 3, title: 'Why Vue is so fun' }
    ]
  }
})
```

```
<blog-post
  v-for="post in posts"
  v-bind:key="post.id"
  v-bind:title="post.title"
></blog-post>
```

Listening to child components event

In children, **child** it will emit an event that will be retrieved by parent

```
export default {  
  methods: {  
    onClickButton (event) {  
      this.$emit('clicked')  
    }  
  }  
}
```

Parent will pass the v-on:clicked to the child

```
<div>  
  <child v-on:clicked="onClickChild"></child>  
</div>
```

And implement the onClickChild method

```
export default {  
  methods: {  
    onClickChild () {  
      console.log('sent from children')  
    }  
  }  
}
```

Passing up event with value

Sometimes we may require communicating back up to the parent. In this case pass it as a second argument of the function

```
export default {
  methods: {
    onClickButton (event) {
      this.$emit('clicked', 'somevalue')
    }
  }
}
```

You may retrieve it directly in the parent, by adding the parameter on the function

```
export default {
  methods: {
    onClickChild (value) {
      console.log(value) // someValue
    }
  }
}
```

Command for projects

vue create api-exe

Cd api-exe

vue add router -> router configuration in Vue JS

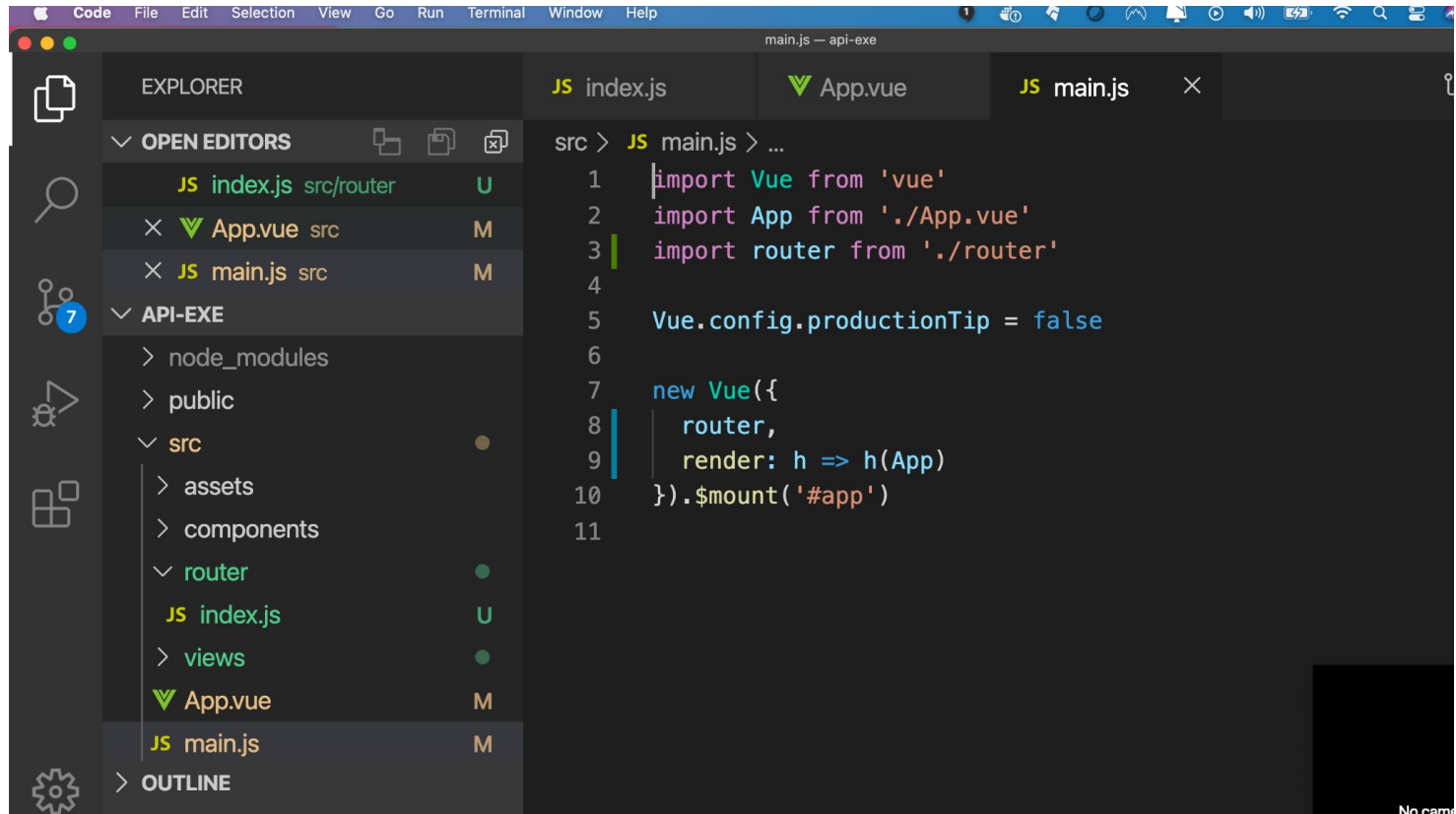
If asked using history mode, select “Y”

Npm run serve

After installation you may run the application, verify that you will have two tabs now, one in home and about

<https://router.vuejs.org/>

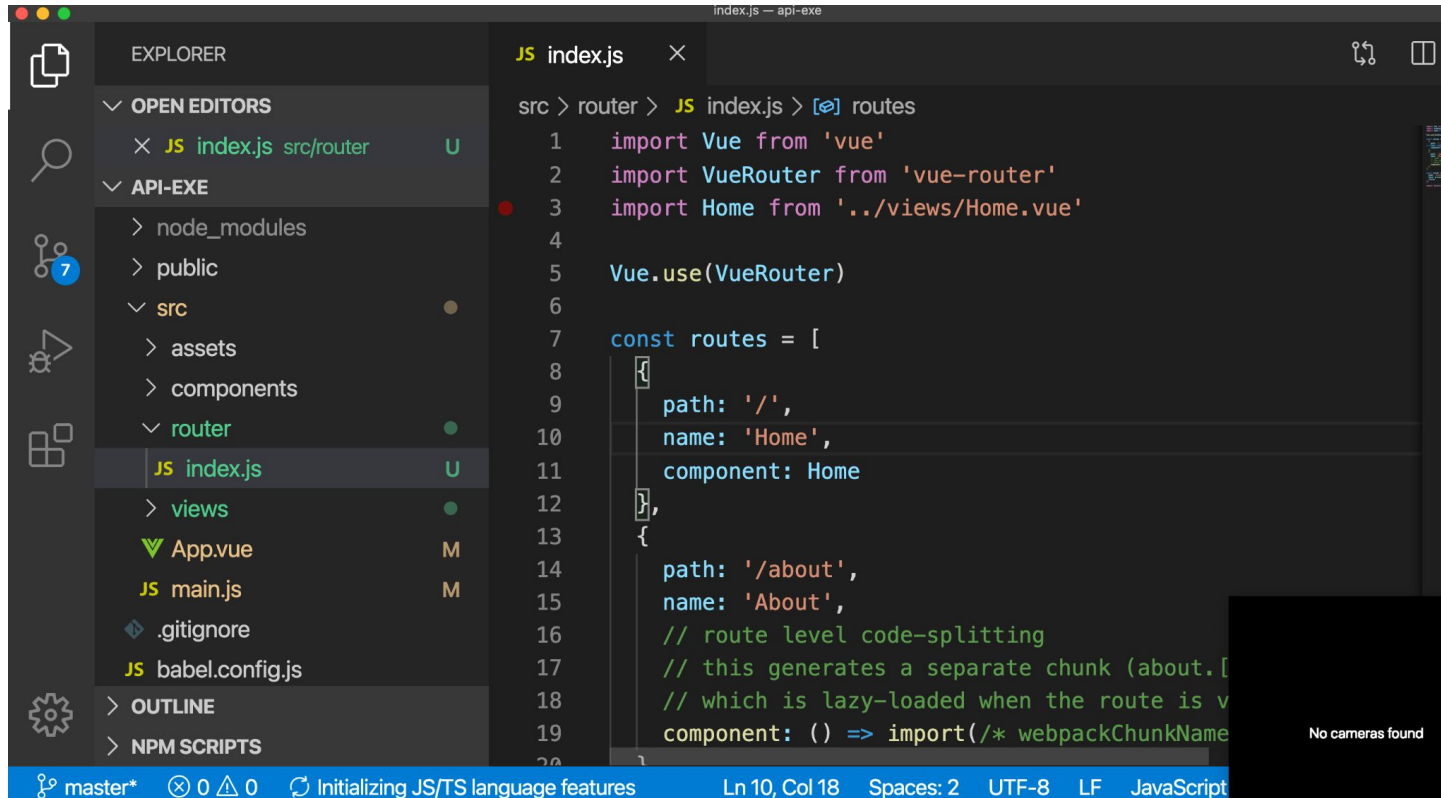
main.js will add router module



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure, including a 'src' directory with subdirectories like 'assets', 'components', 'router', and 'views'. The 'router' directory is expanded, showing 'index.js'. The main editor area displays the content of 'main.js', which includes imports for 'vue', 'App', and 'router', and a Vue instance configuration.

```
src > JS main.js > ...
1  import Vue from 'vue'
2  import App from './App.vue'
3  import router from './router'
4
5  Vue.config.productionTip = false
6
7  new Vue({
8    router,
9    render: h => h(App)
10 }).$mount('#app')
```


router / index.js file will be created



The screenshot shows a Visual Studio Code editor window with the following content:

- EXPLORER:** A file tree on the left showing the project structure. The `router` folder is expanded, and `index.js` is selected.
- JS index.js:** The main editor displays the following code:

```
src > router > JS index.js > routes
1  import Vue from 'vue'
2  import VueRouter from 'vue-router'
3  import Home from '../views/Home.vue'
4
5  Vue.use(VueRouter)
6
7  const routes = [
8    {
9      path: '/',
10     name: 'Home',
11     component: Home
12   },
13   {
14     path: '/about',
15     name: 'About',
16     // route level code-splitting
17     // this generates a separate chunk (about.[hash].js)
18     // which is lazy-loaded when the route is visited
19     component: () => import(/* webpackChunkName: "about" */
20     './views/about.vue')
21   }
22 ]
```
- Status Bar:** Shows the current file is `index.js` at `Ln 10, Col 18`, using `UTF-8` encoding and `LF` line endings.

No cameras found

App.vue will have header

```
src > App.vue > template > div#app
1 <template>
2   <div id="app">
3     <div id="nav">
4       <router-link to="/">Home</router-link> |
5       <router-link to="/about">About</router-link>
6     </div>
7     <router-view/>
8   </div>
9 </template>
10
11 <style>
12 #app {
13   font-family: Avenir, Helvetica, Arial, sans-serif;
14   -webkit-font-smoothing: antialiased;
15   -moz-osx-font-smoothing: grayscale;
16   text-align: center;
17   color: #2c3e50;
```

We add a new View and call it Detail.vue.. It is just a simple page showing following code:

```
<template>
  <div class="detail">
    <h1>This is the detail
page</h1>
  </div>
</template>
```

Add Detail path inside index.js

```
import Detail from '../views/Detail.vue'
```

```
... . . .
```

```
{  
  path: '/detail',  
  name: 'Detail',  
  component: Detail  
}
```

Modify App.vue to include the link to Detail

```
<template>  
  <div id="app">  
    <div id="nav">  
      <router-link to="/">Home</router-link> |  
      <router-link to="/about">About</router-link> |  
      <router-link to="/detail">Detail</router-link>  
    </div>  
    <router-view/>  
  </div>
```

Notes that this will
be replaced by
the page / url

Enter a movie

Search

List of movies

Poster

Film title
Plot
Directors
Actors ..

Vue Router

Vue Router is the official router for [Vue.js](#)

. It deeply integrates with Vue.js core to make building Single Page Applications with Vue.js a breeze. Features include:

- Nested route/view mapping
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects powered by Vue.js' transition system
- Fine-grained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode, with auto-fallback in IE9
- Customizable Scroll Behavior

Create the UI

To create something like the UI above (first page) I will add two more components, Search.vue and List.vue , both are created inside components folder.

Starter code for Search and List component

```
<template>  
  <div>  
  </div>  
</template>
```

```
<script>  
export default {  
  name: 'Search',  
}  
</script>
```

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
```

```
<style scoped>
```

```
</style>
```

HTML for search (template)

```
<template>
  <div class="row">
    <div class="col-md-8 col-sm-12">
      <input type="text" class="form form-control"
        placeholder="Enter movie name">
    </div>
    <div class="col-md-4 col-sm-12">
      <button class="btn btn-primary">
        Search a movie</button>
    </div>
  </div>
</template>
```

API for list (template)

```
<template>
  <div>
    <div class="card p-5 mb-3 bg-light">
      <h2>This is movie 1</h2>
      <p>2003</p>
    </div>
    <div class="card p-5 mb-3 bg-light">
      <h2>This is movie 2</h2>
      <p>2003</p>
    </div>
    <div class="card p-5 mb-3 bg-light">
      <h2>This is movie 3</h2>
      <p>2003</p>
    </div>
  </div>
</template>
```

Home.vue will include both components

```
<template>
  <div class="container">
    <Search />
    <List />

  </div>
</template>
```

```
<script>
  // @ is an alias to /src
  import List from
    "@/components/List.vue";
  import Search from
    "@/components/Search.vue";
  export default {
    name: "Home",
    components: {
      List,
      Search
    }
  };
</script>
```

Home.vue

- Movies info...

```
return {  
  movies: []  
}
```

.. Will pass movie to List

```
<List v-bind:movies=movies />
```

- Will receive user search from child..

```
export default {  
  name: 'List',  
  props: ['movies']  
}
```

-- Search.vue

- Will pass what user search to parent

```
<button class="btn btn-primary"  
  style="width:100%"  
  v-on:click="searchPressed">  
  Search a movie</button>
```

```
searchPressed: function() {
```

```
  this.$emit('searchPressed', this.userSearch)  
}
```

-- **List.vue** component -> will receive movies component from parent

Fetch - get data from API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

```
callApi:function(val) {  
    fetch(`https://www.omdbapi.com/?s=${val}&apikey=87d10179`)  
    .then(response => response.json())  
    .then(data => this.movies = data["Search"]);  
}
```

<http://www.omdbapi.com/?s=Harry&apikey=87d10179>

Add v-for to bring out the data

```
<div>
  <div class="card p-5 mb-3 bg-light" v-for="movie
in movies"
  :key="movie.imdbID">
    <h2>This is movie 1</h2>
    <p>2003</p>
  </div>
</div>
```

Bring out full data including picture (:v-bind)

```
<div>
  <div class="card p-5 mb-3 bg-light" v-for="movie in movies"
    :key="movie.imdbID">
    <div class="row">
      <div class="col-sm-4">
        
      </div>
      <div class="col-sm-8">
        <h2>{{movie.Title}}</h2>
        <p>{{movie.Year}}</p>
      </div>
    </div>
  </div>
</div>
```


Create an onclick on the movie to open detail page..

```
<div class="card p-5 mb-3 bg-light" v-for="movie in movies" :key="movie.imdbID" v-on:click="openDetail(movie)">
```

```
</div>
```

```
methods: {
```

```
  openDetail: function (movie) {
```

```
    console.log(movie);
```

```
  }
```

```
}
```

Programmatic Navigation

Aside from using `<router-link>` to create anchor tags for declarative navigation, we can do this programmatically using the router's instance methods.

```
router.push(location, onComplete?, onAbort?)
```

Note: Inside of a Vue instance, you have access to the router instance as `$router`. You can

Open to detail page programmatically

```
export default {  
  name: 'List',  
  props: ['movies'],  
  methods: {  
    openDetail: function (movie) {  
      console.log(movie)  
      this.$router.push('detail')  
    }  
  }  
}
```

Passing data through path

Our API needs to know the imdbID of selected movie, so update the path to retrieve the imdbID from the url

```
{  
  path: '/detail/:imdbID',  
  name: 'Detail',  
  component: Detail  
}
```

Pass the imdbID through the path using push

```
methods:{
  openDetail:function (movie) {
    console.log (movie)
    this.$router.push(`detail/${movie.imdbID}`)
  }
}
```

You will retrieve later the data using `this.$route.params.imdbID`

Vue Router

Vue Router is the official router for [Vue.js](#)

. It deeply integrates with Vue.js core to make building Single Page Applications with Vue.js a breeze. Features include:

- Nested route/view mapping
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects powered by Vue.js' transition system
- Fine-grained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode, with auto-fallback in IE9
- Customizable Scroll Behavior

Vue router

With `vue router-cli`, to add vue router into your project, you just need to add the following command line:

```
vue add router
```

Setting up routing

```
import Vue from 'vue'  
import VueRouter from 'vue-router'
```

```
Vue.use(VueRouter)
```

```
const router = new VueRouter({  
  mode: 'history',  
  base: process.env.BASE_URL,  
  routes  
})
```

```
export default router
```


Routing and passing parameter

pattern	matched path	\$route.params
/user/:username	/user/evan	<pre>{ username: 'evan' }</pre>
/user/:username/post/:post_id	/user/evan/post/123	<pre>{ username: 'evan', post_id: '123' }</pre>

You may retrieve the parameter using `$route.params.id`

Create route inside routes.js file

```
const routes = [  
  {  
    path: '/',  
    name: 'Main',  
    component: Main  
  },  
  {  
    path: '/detail/:sendDate/:returnDate/:city',  
    name: 'Detail',  
    component: Detail  
  },  
  {  
    path: '/about',  
    name: 'About',  
    component: () => import(/* webpackChunkName: "about" */ '../views/About.vue')  
  },  
  {  
    path: '/car/:id',  
    name: 'Car',  
    component: Car  
  }  
]
```

Inside main.js modify as follow to include route

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

```
Vue.config.productionTip = false
```

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

Retrieving data from params

```
sendData:this.$route.params.sendDate,  
    returnDate:this.$route.params.returnDate,  
    city:this.$route.params.city
```

API call with Axios

```
new Vue({
  el: '#app',
  data () {
    return {
      info: null
    }
  },
  mounted () {
    axios
      .get('https://api.coindesk.com/v1/bpi/currentprice.json')
      .then(response => (this.info = response))
  }
})
```

JS

Fetch API

```
created() {  
  
  // GET request using fetch with set headers const  
  headers = { "Content-Type": "application/json" };  
  fetch("https://api.npms.io/v2/search?q=vue",  
        { headers }) .then(response => response.json())  
    .then(data => (this.totalVuePackages = data.total)); }  
}
```

Import and call the components

```
8
9 <script>
10   import HelloWorld from './components/HelloWorld.vue'
11   export default {
12     name: 'App',
13     components: {HelloWorld},
14     data() {
15       1 <template>
16         2   <div id="app">
17           3     {{msg}}
18           4     <HelloWorld/>
19           5   </div>
20         6
21         7 </template>
22         8
```

Passing data in components (through props)

component, we can include it in the list of props this component accepts, using a `props` option.

```
Vue.component('blog-post', {  
  props: ['title'],  
  template: '<h3>{{ title }}</h3>'  
})
```

JS

A component can have as many props as you'd like and by default, any value can be passed to any

```
<blog-post title="My journey with Vue"></blog-post>  
<blog-post title="Blogging with Vue"></blog-post>  
<blog-post title="Why Vue is so fun"></blog-post>
```

HTML

Vue router

Add vue router into your project using the following command line:

```
vue add router
```

Setting up routing

```
import Vue from 'vue'  
import VueRouter from 'vue-router'
```

```
Vue.use(VueRouter)
```

```
const router = new VueRouter({  
  mode: 'history',  
  base: process.env.BASE_URL,  
  routes  
})
```

```
export default router
```

Create route inside

```
const routes = [  
  {  
    path: '/',  
    name: 'Main',  
    component: Main  
  },  
  {  
    path: '/detail/:sendDate/:returnDate/:city',  
    name: 'Detail',  
    component: Detail  
  },  
  {  
    path: '/about',  
    name: 'About',  
    component: () => import(/* webpackChunkName: "about" */ '../views/About.vue')  
  },  
  {  
    path: '/car/:id',  
    name: 'Car',  
    component: Car  
  }  
]
```

Inside main.js modify as follow to include route

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

```
Vue.config.productionTip = false
```

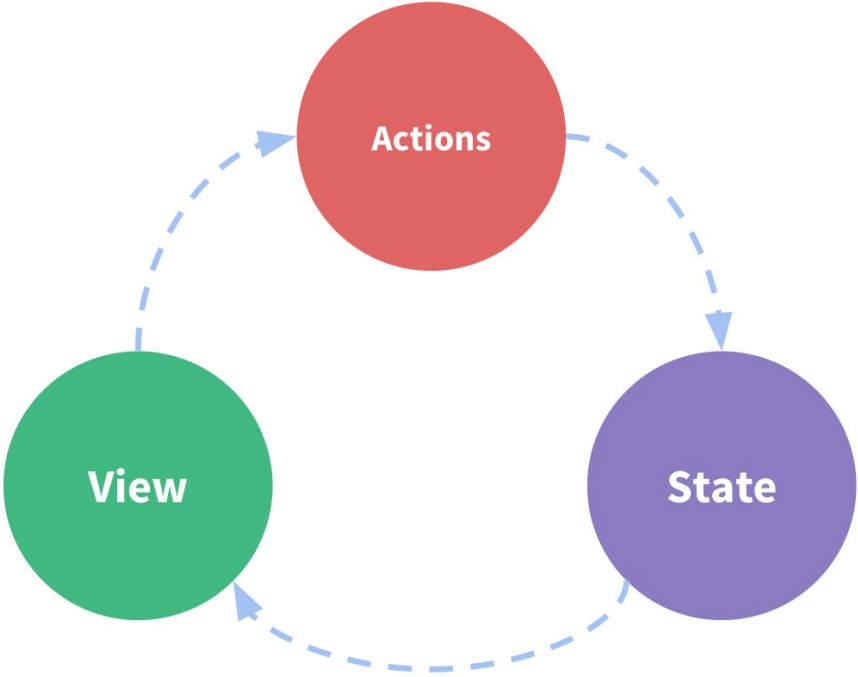
```
new Vue({  
  router,  
  render: h => h(App)  
}).$mount('#app')
```

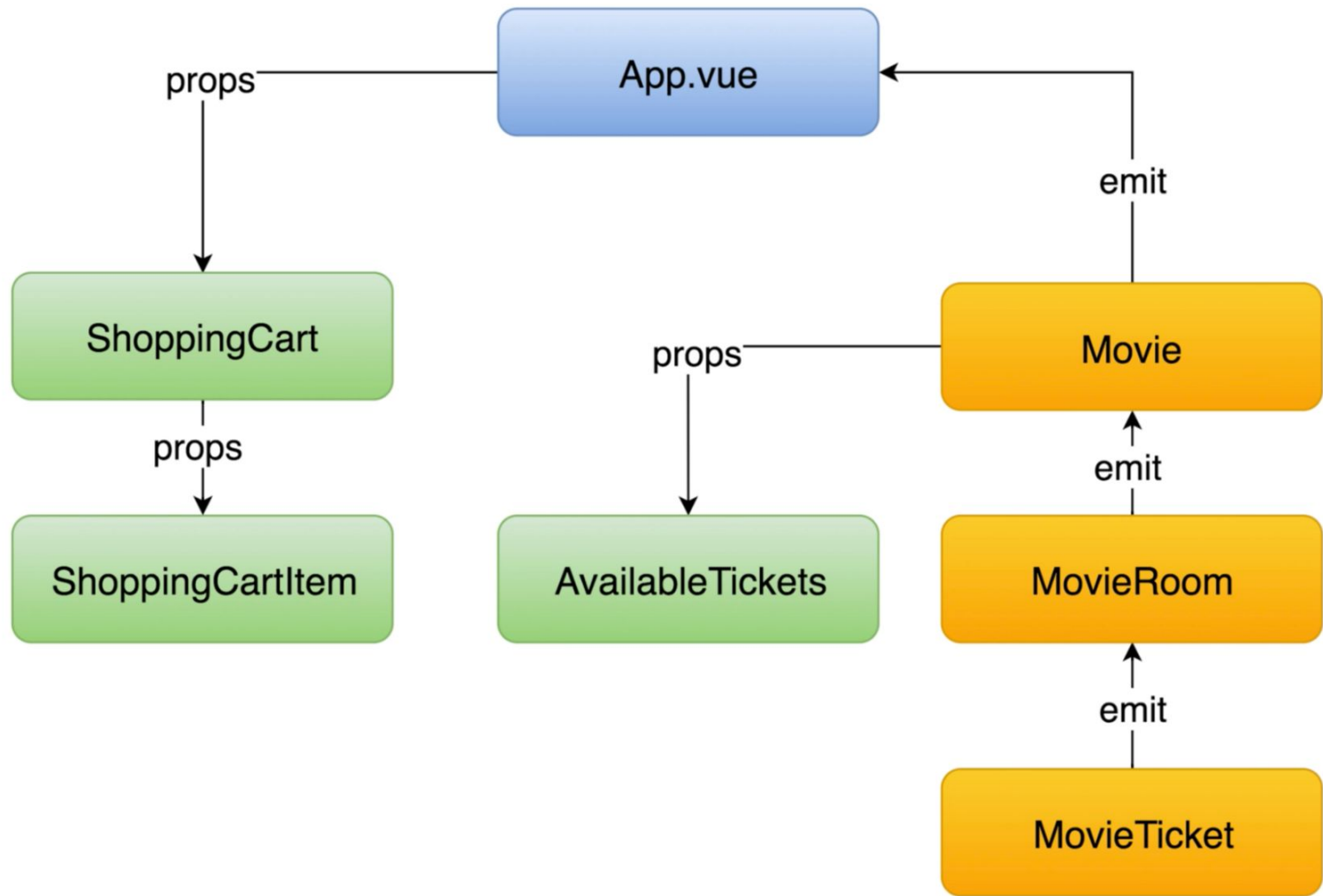
Retrieving data from params

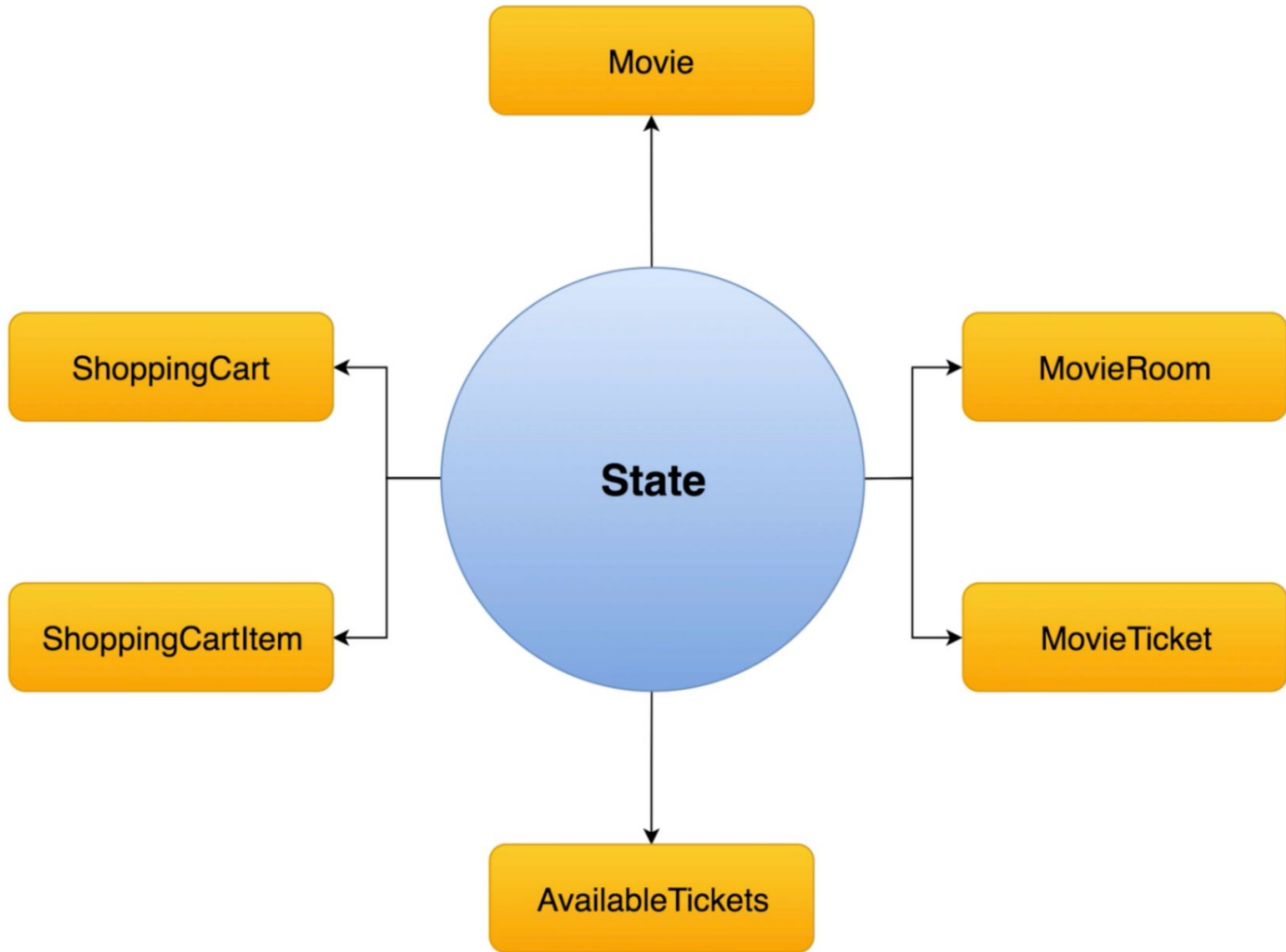
```
sendData:this.$route.params.sendDate,  
    returnDate:this.$route.params.returnDate,  
    city:this.$route.params.city
```

VueJS and Laravel

Part 2 : Introduction to Vuex







What is Vuex

Vuex is a state management pattern + library for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion.

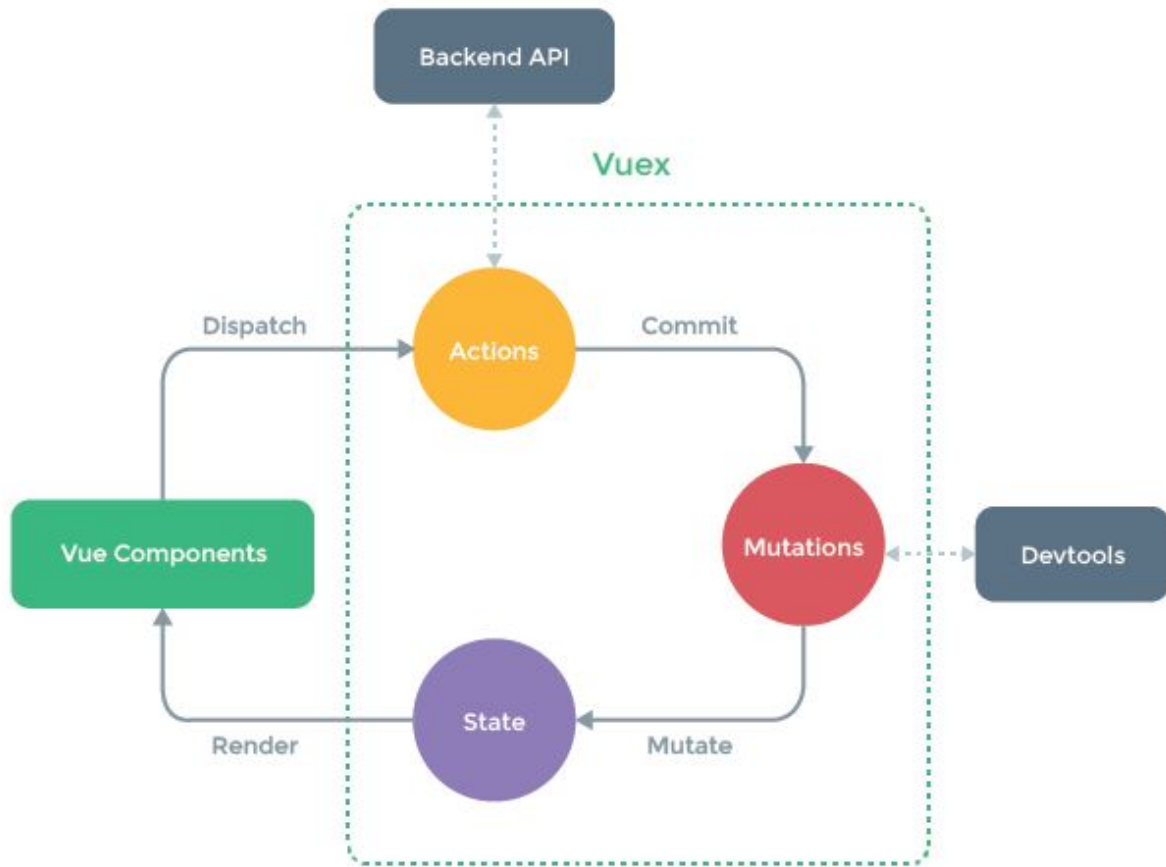
Why Vuex?

However, the simplicity quickly breaks down when we have multiple components that share a common state:

- Multiple views may depend on the same piece of state.
- Actions from different views may need to mutate the same piece of state.

For problem one, passing props can be tedious for deeply nested components, and simply doesn't work for sibling components.

For problem two, we often find ourselves resorting to solutions such as reaching for direct parent/child instance references or trying to mutate and synchronize multiple copies of the state via events. Both of these patterns are brittle and quickly lead to unmaintainable code.



Create a new project (Counter vuex)

- 1) Command line - vue create countervuex
- 2) Use vue2
- 3) cd countervuex
- 4) Add vuex inside the project - npm install vuex
- 5) Open project in ***Visual Studio Code***

Getting started

```
import Vue from 'vue'  
import Vuex from 'vuex'  
  
Vue.use(Vuex)
```

```
npm install vuex --save
```

Code to get started (inside file store/index.js)

```
import Vue from 'vue';  
import Vuex from 'vuex';  
Vue.use(Vuex);  
  
export const store = new Vuex.Store({  
  // store properties  
});
```

Store

At the center of every Vuex application is the store. A "store" is basically a container that holds your application state. There are two things that make a Vuex store different from a plain global object:

1. Vuex stores are reactive. When Vue components retrieve state from it, they will reactively and efficiently update if the store's state changes.
2. You cannot directly mutate the store's state. The only way to change a store's state is by explicitly committing mutations. This ensures every state change leaves a track-able record, and enables tooling that helps us better understand our applications.

#

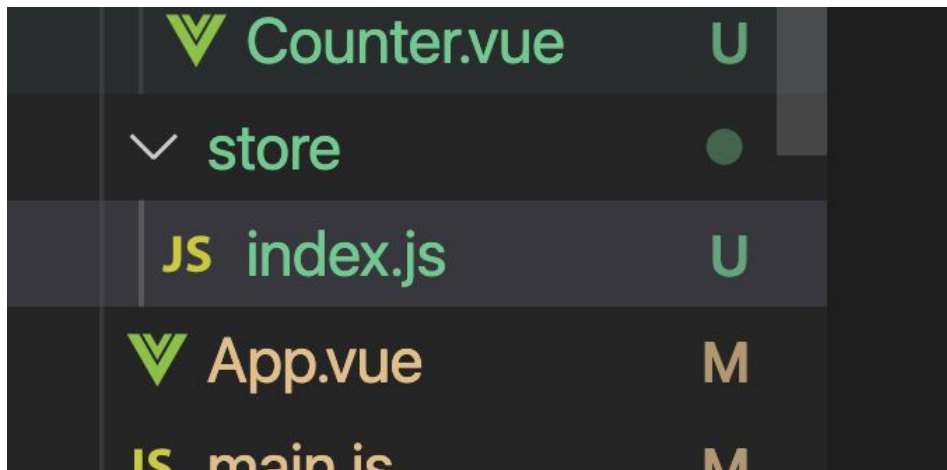
The Vue Store

The Vuex Store is made complete with 4 objects - **state**, **mutations**, **actions**, and **getters**.

- **State** is simply an object that contains the data that needs to be shared within the application.
- **Mutations** are functions responsible in directly mutating store state..
- **Actions** exist to call mutations. Actions are also responsible for performing any or all asynchronous calls prior to committing to mutations.
- **Getters** are primarily used to perform some calculation/manipulation to store state before having that information accessible to components.

Starter code for store

```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)
export default new Vuex.Store({
  state: {
  },
  getters: {
  },
  mutations: {
  },
  actions: {
  }
})
```



Adding Store in main.js (to use it throughout the application)

```
import Vue from 'vue'  
import App from './App.vue'  
import store from './store'  
  
Vue.config.productionTip = false  
  
new Vue({  
  store,  
  render: h => h(App),  
}).$mount('#app')
```

Once added here, you can retrieve the store from any components using `this.$store` or `$store`

We have previously done this... using `router..` (with help from `vue add router`)

Example (1) - Counter app - state

put variables and collections here

```
state: {  
  count: 0,  
  
},
```

<https://vuex.vuejs.org/guide/state.html#single-state-tree>

Calling from components (state)

To call the state
from the
components, you

may use the

Or call it directly from the template

```
<p>{{ $store.state.counter }}</p>
```

following:

Example (1) - Counter app - mutations

put synchronous functions for changing state . e.g. add, edit, delete. Mutations will receive state as parameter by default

```
increment(state) {  
    state.count++  
},  
decrement(state) {  
    state.count--;  
}
```

Example (1) - Counter app - actions

// put asynchronous functions that can call one or more mutation functions. It will have commit as parameter.. Call the mutations from commit (something like transaction in db)

```
actions: {
  increment: ({ commit }) => commit('increment'),
  decrement: ({ commit }) => commit('decrement'),
  incrementAsync: ({ commit }) => {
    setTimeout(() => { commit('increment') }, 1000);
  },
  incrementIfOdd: ({ commit, getters }) => getters.parity === 'odd' ? commit('increment') : false,
}
```

Calling from components (actions)

To call the action from the components :

```
methods:{
  increment:function(){
    store.dispatch('increment');

  },
  decrement:function(){
    store.dispatch('decrement');
  }
}
```

You may also send parameter in the actions

Calling from components (getters)

Created the getters:

```
parity: (state, getters) =>{  
    return state.counter % 2 == 0 ? "even" : "odd";  
}
```

To call getters from the component you can use the following code:

```
computed: {  
  
doneTodosCount () {  
    return this.$store.getters.parity  
}  
}
```

Or similar to state you may call it directly from the code:

```
<p>{{ $store.state.counter }} . It is a {{ $store.getters.parity }} number</p>
```

mapState

When a component needs to make use of multiple store state properties or getters, declaring all these computed properties can get repetitive and verbose. To deal with this we can make use of the `mapState` helper which generates computed getter functions for us:

```
computed: {  
  ...mapState([  
    'count'  
  ]),  
}
```

mapGetters

The `mapGetters` helper simply maps store getters to local computed properties:

```
...mapGetters([  
  'parity'  
])
```

```
computed:{
  ...mapState([
    'counter'
  ]),
  ...mapGetters([
    'parity'
  ])
},
```

<p>{{counter}} . It is a {{parity}} number</p>

mapAction

`mapActions` helper which maps component methods to `store.dispatch` calls (requires root store injection):

```
methods: mapActions([
  'increment',
  'decrement',
  'incrementIfOdd',
  'incrementAsync'
])
```

<https://vuex.vuejs.org/api/#mapactions>

Source code for Vuex Counter

<https://github.com/wanmuz86/vuex-counter>

Shopping cart exercise

1. `vue create shoppingcart`
2. `cd shoppingcart`
3. `npm install vuex`
4. `npm run serve`
5. Create two components : `ShoppingList`, `Product`
6. Follow the two next slides to evaluate normal way of retrieving data
7. Create store and add it in `main.js` as per previous exercise.

Code to retrieve a product

```
/**
 * Mocking client-server processing
 */
const _products = [
  { 'id': 1, 'title': 'iPad 4 Mini', 'price': 500.01, 'inventory': 2 },
  { 'id': 2, 'title': 'H&M T-Shirt White', 'price': 10.99, 'inventory': 10 },
  { 'id': 3, 'title': 'Charli XCX - Sucker CD', 'price': 19.99, 'inventory': 5 }
]
```


Code to retrieve a product (2)

```
export default {  
  getProducts (cb) {  
    setTimeout(() => cb(_products), 100)  
  },  
  buyProducts (products, cb, errorCallback) {  
    setTimeout(() => {  
      // simulate random checkout failure.  
      (Math.random() > 0.5 || navigator.webdriver)  
        ? cb()  
        : errorCallback()  
    }, 100)  
  }  
}
```

Bring it out normal way

```
<ul>
  <li v-for="product in products" :key="product.id">{{product.title}} - {{product.price}} -
  {{product.inventory}}</li>
</ul>
```

```
import shop from '@api/shop';
export default {
  name: 'Product',
  created() {
    shop.getProducts(products => this.products = products)
  },
  data() {
    return {
      products: []
    }
  }
}
```

Example (2) - Shopping List app - State

```
products: []
```

Example (2) - Shopping List app - mutations

```
setProducts(state, products) {  
    state.products = products  
},
```

```
decrementProductInventory(state, product) {  
    product.inventory--  
},
```

In this example, we pass a parameters products. Inside the mutations...

You may do this

Example (2) - Shopping List app - actions

```
actions: {  
  fetchProducts({commit}) {  
    return new Promise((resolve, reject)=>{  
      shop.getProducts(products => {  
        commit('setProducts',products);  
        resolve()  
      })  
    })  
  }  
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Our Product.vue

```
export default {
  name: 'Product',
  created() {
    this.fetchProducts()
      .then(() => console.log(this.$store.state.products));
  },
  computed: {
    ...mapState([
      'products'
    ])
  },
  methods: {
    ...mapActions([
```

Updated code for shopping list

<https://github.com/wanmuz86/vuex-shoppinglist>

Recap of our project

API
(Where we get
our information)

Store/index.js

- state
- mutations
- getters
- actions

Product.vue
(component)

ShoppingList.vue
(component)

Recap of each component in Vuex

Store - Is a file which part of Vuex architecture to centralize our state management and make everything in one file.

State - variable that's being used in our application

Mutation - the function that is used to change the value of state. This cannot be changed directly by components.

Getters - To get the transformed value of state, eg: either it is odd or even, either item is available or not

Actions - is a method to invoke the state , eg: setters, getters, addToCart

Eg: addToCart action

- From Product.vue, I will call addToCart actions..
- addToCard will call **pushProductToCart mutators**
- **pushProductToCart mutator** will add the item and it's quantity inside cart state

In reality, when user press addToCart ... what will happen

- We check first is productsAvailable (inside products API will have inventory information)
- If productsAvailable... then I check If item is already inside the cart..
 - If item is inside the cart... I will **change the quantity** of the item in the **cart**
 - If not... then I will **add the item** in **cart..** (push) -> Our current code
- Since I have already add the product in cart, I will **decrease the inventory** in **product...**

Example 2 - Shopping List app - actions

```
addProductToCart({state, getters, commit}, product) {  
  if (getters.productIsInStock(product)) {  
  
    const cartItem = state.cart.find(item => item.id == product.id)  
    if (!cartItem) {  
      commit('pushProductToCart', product.id)  
    }  
    else {  
      commit('incrementItemQuantity', cartItem)  
    }  
    commit('decrementProductInventory', product)  
  }  
}
```

Update our mutations to add the two new mutations

```
incrementItemQuantity(state, cartItem) {  
    cartItem.quantity++;  
},  
decrementProductInventory(state, product) {  
    product.inventory--;  
}
```

Problem

The problem now is, we can still keep reducing the items if the inventory/stock is no longer there..

What we are going to do, we are going to disable the button if the products are no longer available in stock.

Since we are getting **the attribute** of our state, and process information from it, we will use getter for this.

Add a new getters to check if product is in stock

```
availableProducts(state, getters) {  
    return state.products.filter(product =>  
product.inventory > 0)  
},  
  
productIsInStock() {  
    return (product) =>{  
        return product.inventory > 0  
    }  
  
}
```

Now you will call the getters using v-bind to disabled the button if it is not available...

```
<ul>
```

```
  <li v-for="product in products"
```

```
    :key="product.id">{{product.title}}
```

```
      - {{product.price}} - {{product.inventory}}
```

```
      <button v-on:click="addProductToCart (product) "
```

```
        v-bind:disabled="!productIsInStock (product) ">Add to
```

```
Cart</button></li>
```

```
</ul>
```


Don't forget to add mapGetters in our Product.vue Component

```
import {mapActions, mapState, mapGetters} from 'vuex'

export default {
  name: 'Product',
  created(){
    this.fetchProducts()
      .then(()=>console.log(this.$store.state.products));

  },
  computed:{
    ...mapState([
      'products'
    ]),
    ...mapGetters([
      'productIsInStock'
    ])
  },
```

Managing checkout

We will create a new actions which is called checkout... Checkout will call the buyProducts API and perform two things when it happen

If successful, - **empty cart**, change **checkoutstatus** to successful

If unsuccessful, -change **checkoutstatus** status to unsuccessful

Call the checkout action when the button is pressed in Checkout component...

Show the checkout status as well

1. Create new state - checkoutStatus

```
export default new Vuex.Store({  
  state: {  
    products: [],  
    cart: [],  
    checkoutStatus: null  
  },
```

2. Create new mutators (emptyCart & setCheckoutStatus)

```
emptyCart (state) {  
    state.cart = []  
},  
setCheckoutStatus (state, status) {  
    state.checkoutStatus = status  
}
```

3) Create buyProducts actions that will call the API

```
buyProducts({state, commit}, products) {  
  shop.buyProducts(state.carts, ()=>{  
    // Call API, if successful  
    commit('emptyCart');  
    commit('setCheckoutStatus', true);  
  
  }, ()=>{  
    // If error will go here  
    commit('setCheckoutStatus', false);  
  })  
}
```

Add mapActions and call it on a button

```
import { mapState, mapGetters, mapActions } from 'vuex'
export default {
  name: 'ShoppingList',
  computed: {
    ...mapState([
      'cart'
    ]),
    ...mapGetters([
      'cartProducts',
      'cartTotal'
    ])
  }
}
```

Call the buyProducts actions on a button

```
<p><button
```

```
v-on:click="buyProducts">Checkout</button></p>
```

Add a getters to add human readable status

```
checkoutStatus(state) {  
    if (state.checkoutStatus !== null) {  
        return state.checkoutStatus ? "Successfully  
checkout" : "Something went wrong";  
    }  
    return null  
}
```


Now you can call the getters in the component

```
<p>My Cart</p>
  <ul>
    <li v-for="item in cartProducts" :key="item.title">{{item.title}} - {{item.price}} -
{{item.quantity}}</li>
  </ul>
  <p>Total Price : {{cartTotal}}</p>
  <p><button v-on:click="buyProducts">Checkout</button></p>
  <p v-if="checkoutStatus">{{checkoutStatus}}</p>
```

```
...mapGetters([
  'cartProducts',
  'cartTotal',
  'checkoutStatus'
])
```

Example (2) - Shopping List app - State (Cart)

```
cart: [],
```

```
checkoutStatus:null
```

Example 2 - Shopping List app - mutations

```
pushProductToCart (state, productId) {
    state.cart.push({
        id:productId,
        quantity:1
    })
},
incrementItemQuantity(state, cartItem){
    cartItem.quantity++
},
setCheckoutStatus (state, status) {
    state.checkoutStatus = status
},
emptyCart (state) {
    state.cart = []
}
```

For the first part of exercise we used this actions first (simplified)

```
addProductToCart ({commit}, product) {  
    commit ('pushProductToCart', product.id);  
}
```

Inside our Product.vue (add the mapActions and v-on:click)

```
<li v-for="product in products"  
:key="product.id">{{product.title}} - {{product.price}} -  
{{product.inventory}} <button  
v-on:click="addProductToCart (product) ">Add to  
Cart</button></li>
```

```
...mapActions([  
  'fetchProducts',  
  'addProductToCart'  
])
```

Temporary Cart List (1)

```
<template>
  <div>
    <p>My Cart</p>
    <ul>
      <li v-for="item in cart"
:key="item.id">{{item.id}} -
{{item.quantity}}</li>
    </ul>
  </div>
</template>
```

```
<script>
import { mapState } from 'vuex'
export default {
  name: 'ShoppingList',
  computed:{
    ...mapState([
      'cart'
    ])
  },
}
</script>
```

Example 2 - Shopping List app - Getters

```
cartProducts(state) {  
  return state.cart.map(cartItem=>{  
    const product = state.products.find(product=> product.id === cartItem.id)  
    return {  
      title:product.title,  
      price:product.price,  
      quantity:cartItem.quantity  
    }  
  })  
},  
cartTotal(state, getters) {  
  return getters.cartProducts.reduce((total, product) => total + product.price * product.quantity, 0)  
},
```

```
<template>
  <div>
    <p>My Cart</p>
    <ul>
      <li v-for="item in cartProducts" :key="item.title">{{item.title}} - {{item.price}} -
{{item.quantity}}</li>
    </ul>
    <p>Total Price : {{cartTotal}}</p>
  </div>
</template>
```



```
<script>
import { mapState,mapGetters } from 'vuex'
export default {
  name: 'ShoppingList',
  computed:{
    ...mapState([
      'cart'
    ]),
    ...mapGetters([
      'cartProducts',
      'cartTotal'
    ])
  },
}
</script>
```

```
<style>
```

```
</style>
```

Product List

```
<p v-if="loading">Loading...</p>
<h1>Product List</h1>
<ul>
  <li v-for="product in products" :key="product.id">
    {{product.title}} - {{product.price | currency}} - {{product.inventory}}
    <button
      :disabled="!productIsInStock(product)"
      v-on:click="addProductToCart(product)">Add to cart</button>
  </li>
</ul>
</div>
```

Product List - Data & Computed

```
data() {  
  return {  
    loading:false  
  }  
},  
computed:{  
  ...mapState({  
    products:state=>state.products  
  } ),  
  ...mapGetters({  
    productIsInStock:'productIsInStock'  
  })  
}
```

Product List - Created & Methods

```
created(){
  this.loading = true

  this.fetchProducts()
  .then(()=> this.loading = false)
},
methods: {
  ...mapActions({
    fetchProducts:'fetchProducts',
    addProductToCart:'addProductToCart'
  })
  // addProductToCart (product) {
  //   this.$store.dispatch('addProductToCart',product)
  // }
}
```

Shopping Cart - Computed

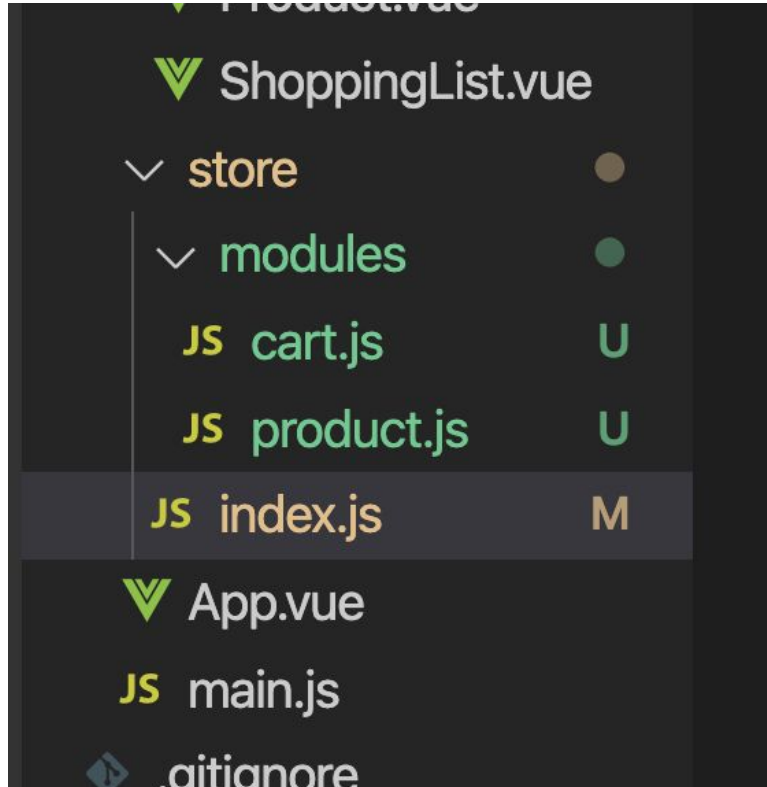
```
computed: {  
  
  ...mapGetters({  
    products: 'cartProducts',  
    total: 'cartTotal'  
  }),  
  ...mapState({  
    checkoutStatus: 'checkoutStatus'  
  }),  
},  
methods: {  
  ...mapActions(['checkout'])  
},
```

Suggested application structure

```
sh
├─ index.html
├─ main.js
├─ api
│  └─ ... # abstractions for making API requests
├─ components
│  └─ App.vue
│  └─ ...
└─ store
   └─ index.js      # where we assemble modules and export the store
   └─ actions.js   # root actions
   └─ mutations.js # root mutations
   └─ modules
      └─ cart.js    # cart module
      └─ products.js # products module
```

<https://vuex.vuejs.org/guide/structure.html>

Create new folder modules and create two files inside it



Get the boilerplate and add it in shop.js and

product.js

```
export default ({
```

```
  state: {
```

```
  },
```

```
  getters: {
```

```
},
```

```
  mutations: {
```

```
},
```

```
  actions: {
```


Product module

- State - products
- Getters - productsInStock
- Mutations - setProducts, decrementProductInventory
- Actions: fetchProducts

Refer directly here:

<https://github.com/wanmuz86/vuex-shoppinglist/tree/master/src/store>

Cart module

- State - cart, checkoutStatus
- Getters - cartProducts, cartTotal, checkoutStatus
- Mutations - pushProductToCart, incrementItemQuantity, emptyCart, setCheckoutStatus
- Actions: addProductToCart, buyProducts,

Refer directly here:

<https://github.com/wanmuz86/vuex-shoppinglist/tree/master/src/store>

Set namespace true so we can call it from our component

```
import shop from '@api/shop';  
  
export default {  
  namespace: 'true',  
  state: {  
    cart: [],  
    checkoutStatus: null  
  },  
},
```

Add modules inside index.js and refer to store and product

```
export default new Vuex.Store({  
  modules:{  
    cart,  
    product  
  },
```

** cart and product here refer to the filename..

From the UI, you will retrieve the state like this

```
{  
  ...mapState({  
    product: state => state.product.products  
  })  
},
```

- The bolded part, is the module name (file name) and the highlighted part is the state..

From the UI, you will retrieve the state like this

```
computed: {
```

```
  ...mapState ({
```

```
    cartProducts: state => state.cart.cart
```

```
  }
```

```
),
```

- The bolded part, is the module name (file name) and the highlighted part is the state..

Getters..

For getters, you are going to specify the module name as a parameter of `mapGetters`, for example:

```
...mapGetters('product', [  
  'productIsInStock'  
])  
,
```

product.js

```
...mapGetters('cart', [  
  'cartProducts',  
  'cartTotal',  
  'checkoutStatus'  
])  
,
```

cart.js

Actions (example for product.vue)

```
methods: {  
  ...mapActions({  
    fetchProducts: "product/fetchProducts",  
    addProductToCart: "cart/addProductToCart"  
  })  
}
```

For mapActions.. We link fetchProducts method with
bolded means the module and highlighted is the actions

Actions (example for ShoppingList.vue)

```
methods: {  
  ...mapActions({  
    fetchProducts: "product/fetchProducts",  
    addProductToCart: "cart/addProductToCart"  
  })  
}
```

For mapActions.. We link fetchProducts method with
bolded means the module and highlighted is the actions

Actions (example in Cart)

```
methods: {  
  ...mapActions({  
    buyProducts: 'cart/buyProducts'  
  })  
}
```

For mapActions.. We link buyProducts method with bolded means the module name and highlighted is the actions name inside the module.

Now , we are going to change some of the items inside our module (store for cart and store for product) because some of the modules, are calling each other.

Add Product To Cart will call decrementProductInventory' mutators.. Add the bolded statement

```
addProductToCart({ state, commit }, product) {  
  // Find the item in the cart  
  const cartItem = state.cart.find(item => item.id === product.id)  
  if (!cartItem) {  
    commit('pushProductToCart', product.id);  
  }  
  else {  
    commit('incrementItemQuantity', cartItem);  
  }  
  commit('product/decrementProductInventory', product, {root:true});  
},
```

root:True means refer from index.js, and looks for product module..

In cartProducts the problem is state.products which is referring to the state products in product module

```
cartProducts(state, getter, rootState) {  
    // For each item in the cart  
    return state.cart.map(cartItem => {  
        // I will look for the matching product in products state  
        const product = rootState.product.products.find(product =>  
product.id === cartItem.id)  
        // Then I will return the product information  
        return {  
            title: product.title,  
            price: product.price,  
            quantity: cartItem.quantity  
        }  
    }  
}
```

Use rootstate to get state from other modules

```
getters: {  
  cartProducts(state, getters, rootState) {  
    return state.cart.map(cartItem=>{  
      const product =  
rootState.products.products.find(product=> product.id ===  
cartItem.id)  
  
      return {  
        title:product.title,  
        price:product.price,  
        quantity:cartItem.quantity
```

Use rootgetters to get getters from other module

```
actions: {
  addProductToCart({state, getters, commit, rootState, rootGetters}, product) {
    if (rootGetters['products/productIsInStock'](product)) {

      const cartItem = state.cart.find(item => item.id == product.id)
      if (!cartItem) {
        commit('pushProductToCart', product.id)
      }
      else {
        commit('incrementItemQuantity', cartItem)
      }
      commit('decrementProductInventory', product)
    }
  },
}
```

In .vue file specify the module name

```
...mapActions({  
  fetchProducts: 'products/fetchProducts',  
  addProductToCart: 'cart/addProductToCart'  
})
```

```
computed: {  
  ...mapState({  
    products: state => state.products.products  
  } ),  
},
```


Summary

- Making our store modular.
- In real project, each data will be added inside it's own module
 - You will normally create the module and follow the given boilerplate, (state, mutators, actions, getters) for each of the state.
- From to time, each of the store will depend on each other.., in that case we need to refer back to module..
 - **rootState** - getters, actions [refer reference]
 - For mutation, you will specify the module name, and set {root:true}

Summary

- From the vue..
 - When we call state, actions and getters, we specify from with modules it come from and use `{}` `mapState` instead `[]` `mapState`. These are the way to refer to modules from each of it:

```
...mapState({
  cartProducts: state => state.cart.cart,
})
...mapGetters('cart', [
  'cartProducts',
  'cartTotal',
  'checkoutStatus'
])
methods: {
  ...mapActions({
    buyProducts: 'cart/buyProducts'
  })
}
```

Explanation on difference {} and []

```
mapAction([\n  'buyProducts'\n])
```

I will link the **buyProducts** method in this file to **buyProducts** method in the store..

```
mapAction({\n  buyProducts: 'cart/buyProducts'\n})
```

VueJS and Laravel

Part 3 : ES6 Recap

let

let is a new feature introduced in ES2015 and it's essentially a block scoped version of var . Its scope is limited to the block, statement or expression where it's defined, and all the contained inner blocks.

```
function checkScope() {  
  
  'use strict';  
  
  let i = 'function scope';  
  
  if (true) {  
    let i = 'block scope';  
    console.log('Block scope i is: ', i);  
  }  
  
  console.log('Function scope i is: ', i);  
  
  return i;  
}
```

const

Variables declared with `var` or `let` can be changed later on in the program, and reassigned. Once a `const` is initialized, its value can never be changed again, and it can't be reassigned to a different value.

You will use `let` when you want the variable to change, and `const` when you want the variable to remain constant.

When you use `const`, to conform to common practices, meaning constants should be in all caps.

Arrow functions

In JavaScript, we often don't need to name our functions, especially when passing a function as an argument to another function. Instead, we create inline functions. We don't need to name these functions because we do not reuse them anywhere else.

```
const myFunc = function() {  
  const myVar = "value";  
  return myVar;  
}
```

```
const myFunc = () => {  
  const myVar = "value";  
  return myVar;  
}
```

Data types in Programming

Data types	Example	Operation
String	" " , ' ' . `Hello \${name}`	+
Number	4,4.2,3.3,3.3333	+,-,*,/,%,++,-- , parseInt, parseInteger
Boolean	true, false	&& , , ! , !=
Array - List	[] , number to assess	Push,pop,length, splice, slice
Object / Dictionary / Map /Dict	{}, key to asses, or dot notation	User["name"]

Declarative programming

A programming paradigm that expresses the logic of a computation without describing its control flow. (***What to do***)

The benefit is that declarative style reduces complexity and makes your code easier to read and understand.

Imperative programming

A programming paradigm that uses statements that change a program's state.
(How to do)

Your code focuses on creating **statements that change program states** by creating algorithms that tell the computer **how to do things**.

It closely relates to how hardware works. Typically your code will make use of **conditional statements, loops** and **class inheritance**.

```
Var numbers = [0,1,2,3,4,5]
```

```
Var searchNumber = 3
```

```
numbers.filter (val=>{
```

```
return val != searchNumber
```

```
})
```

```
Var numbers = [0,1,2,3,4,5]
```

```
Var searchNumber = 3
```

```
Var answers = []
```

```
For (var i = 0; i < numbers.length; i++){
```

```
  If (numbers[i] != searchNumber){
```

```
    answers.push(i)
```

```
  }
```

```
}
```

Sample Code: Declarative Programming

```
var arr = [1, 2, 3, 4, 5],  
arr2 = arr.map(function(v, i){ return v*2 })  
console.log('b', arr2)
```

Sample Code: Imperative programming

```
var arr = [1, 2, 3, 4, 5],  
    arr2 = []  
for (var i=0; i<arr.length; i++) {  
    arr2[i] = arr[i]*2  
}  
console.log('a', arr2)
```

Reference to codepen

<https://codepen.io/wanmuz86/pen/abJXOdq?editors=0012>

.map()

The `map()` method is used to apply a function on every element in an array. A new array is then returned.

.map() format and explanation

```
let newArr = oldArr.map((val, index, arr) =>
{
  // return element to new Array
});
```

- `newArr` — the new array that is returned
- `oldArr` — the array to run the map function on
- `val` — the current value being processed
- `index` — the current index of the value being processed
- `arr` — the original array

Example : map()

```
let arr = [1,2,3,4];  
let plus5 = arr.map((val, i, arr) => {  
  return val + 5;  
})  
console.log(plus5)
```

Another Example : map()

```
arr = [1,2,3,4];  
let newArr = arr.map((val, i, arr) => {  
  return {  
    value: val,  
    index: i  
  };  
});  
console.log(newArr)
```

.filter

The `filter()` method returns a new array created from all elements that pass a certain test performed on an original array.

.filter syntax and explanation

```
let newArr = oldArr.filter(callback);
```

The `callback` function can take three arguments:

- `element` — the current element of the array
- `index` — the current index of the value being processed
- `arr` — the original array

Example 1: .filter()

```
let arr = [1,2,3,4,5,6];  
  
let even = arr.filter(val => {  
  return val % 2 === 0;  
});  
  
console.log(even)
```

.reduce

The `reduce()` method is used to apply a function to each element in the array to reduce the array to a single value.

.reduce syntax and explanation

```
let result = arr.reduce(callback);  
// Optionally, you can specify an initial value  
let result = arr.reduce(callback, initialValue);
```

Example 1: .reduce()

```
let arr = [1,2,3,4];  
  
let sum = arr.reduce((acc, val) => {  
  return acc + val;  
});  
  
console.log(sum)
```


Example 2: .reduce() with initial value

```
let arr = [1,2,3,4];

let sum = arr.reduce((acc, val) => {

  return acc + val;

}, 100);
console.log(sum)
```

Ternary operators

(if condition) ? (true statement) : (false statement)

Example

```
val % 2 == 0 ? val*2 : val
```

Map, filter, reduce Exercise

- 1) Create a function that will double the even numbers and leave the odd numbers the same.
- 2) Given an array of objects as follows, return the countries where population more than 500,000,000.

```
let data = [  
  {  
    country: 'China',  
    population: 1409517397,  
  },  
  {  
    country: 'India',  
    population: 1339180127,  
  },  
  {  
    country: 'USA',  
    population: 324459463,  
  },  
  {  
    country: 'Indonesia',  
    population: 263991379,  
  }  
]
```

Map, filter, reduce Exercise

3) Based on the previous array of object, how would you sum up the population of every country except China?

Map, filter, reduce Exercise.

```
data = [  
  {  
    name: 'Butters',  
    age: 3,  
    type: 'dog'  
  },  
  {  
    name: 'Lizzy',  
    age: 6,  
    type: 'dog'  
  },  
  {  
    name: 'Red',  
    age: 1,  
    type: 'cat'  
  },  
  {  
    name: 'Joey',  
    age: 3,  
    type: 'dog'  
  },  
];
```

1. Select only the dogs
2. Translate their ages into dog years (multiply them by seven)
3. Sum the results

1) Even number function

```
arr = [1,2,3,4];  
let newArr = arr.map((v,i,a) => {  
  if (v % 2 === 0){  
    return v * 2;  
  } else {  
    return v;  
  }  
});  
console.log(newArr)
```

2) Filter based on number of population

```
let cities = data.filter(val => {  
  
    return val.population > 500000000;  
  
});
```

3) Sum of country except China

```
let sum = data.reduce((acc, val) => {  
  
  return val.country == 'China' ? acc :  
  acc + val.population;  
  
}, 0);
```


VueJS and Laravel

Web Development
with Laravel
Framework

Syllabus

Tools and Environments

Code Editor

SQL Editor

Database

Eloquent

Migration

Relationship

Collections

Validation

Middleware

Define Middleware

Register Middleware

Others

Using Packages

Helpers

Validation Trait

Display Errors

Form Request Validation

Custom Error Messages

Available Rules

Multilingual Validation Messages

Security

Authentication

Authorization

Fundamental

Development Environment

Routing

Controllers

Request

Response

Views

Blade Templates

Laravel



Developer(s)	Taylor Otwell
Initial release	June 2011; 6 years ago ^[1]
Stable release	5.4.0 ^[2] / January 24, 2017; 5 months ago
Repository	github.com/laravel/framework
Development status	Active
Written in	PHP 5 ^[a]
Operating system	Cross-platform
Type	Web framework
License	MIT License
Website	laravel.com

Tools & Environments

Code Editor

1. Sublime Text 3 or
2. Atom

SQL Editor

1. Sequel Pro (Mac Users) or
2. SQLYog Community (Windows) or
3. HeidiSQL.
4. PHPMyAdmin? Too slow, use any desktop based application :)

Tools & Environments

Development Environment

1. PHP7
2. RDBMS - MySQL, PostgreSQL, Oracle (require external package), SQLite
3. Web Server - Nginx / Apache
4. Cache Manager - Redis / Memcached [OPTIONAL]
5. Dependency Manager
 - a. For PHP - Composer ,
 - b. For Front End Development - NPM (NodeJS Package Manager) [OPTIONAL]

To have mandatory development environment:

1. Windows users, head to Laragon(<https://laragon.org/>) and download the full apache package and install it. That's it. :)
2. For Mac users, my preference to use Homebrew(<https://brew.sh/>) to get all the development environment.
3. For Linux users, you know what to do. :)

Create New App

Create a Laravel Project

Directory Permission

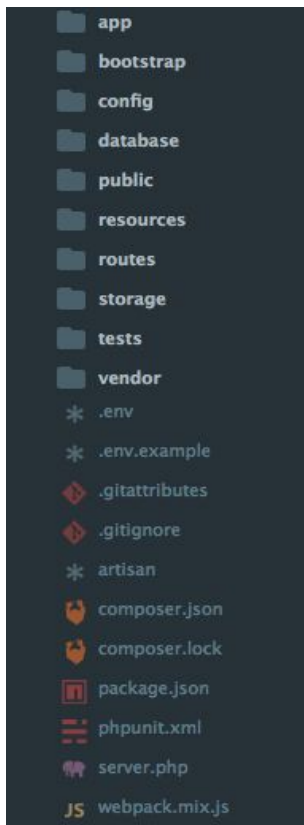
Create a Database for the Project

Configure Database Connection in .env

For Laragon(Windows) users, you don't need to worry about this.

For Mac / Linux users, this is optional if you already install Laravel Installer.

The Structure



The default Laravel application structure is intended to provide a great starting point for both large and small applications. Of course, you are free to organize your application however you like. Laravel imposes almost no restrictions on where any given class is located - as long as Composer can autoload the class.

<https://laravel.com/docs/8.x/structure>

Artisan Console

```
Laravel Framework 5.4.28

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
  --ansi               Force ANSI output
  --no-ansi            Disable ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]         The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2

Available commands:
  clear-compiled  Remove the compiled class file
  down           Put the application into maintenance mode
  env           Display the current framework environment
  help         Displays help for a command
  inspire      Display an inspiring quote
  list        Lists commands
  migrate     Run the database migrations
  optimize    Optimize the framework for better performance
  serve      Serve the application on the PHP development server
  tinker     Interact with your application
  up        Bring the application out of maintenance mode
  app
  app:name  Set the application namespace
  auth
  auth:clear-resets  Flush expired password reset tokens
```

Artisan is the command-line interface included with Laravel. It provides a number of helpful commands that can assist you while you build your application. To view a list of all available Artisan commands, you may use the `list` command:

```
php artisan list
```

<https://laravel.com/docs/5.4/artisan>

Authentication (Quickstart)

Create a Login, Register and Forget Password
Page

[https://laravel.com/docs/5.4/authentication
#authentication-quickstart](https://laravel.com/docs/5.4/authentication#authentication-quickstart)

Exercise

Create 5 projects with following name

1. a100
2. a200
3. a300
4. a400
5. a500

p/s: don't forget to setup database connection(`.env`), run migration scripts and make authentication scaffold.

Fundamental

Routing

Controllers

Views

Blade Templates

Request

Response

Routing



All Laravel routes are defined in your route files, which are located in the `routes` directory. These files are automatically loaded by the framework. The `routes/web.php` file defines routes that are for your web interface. These routes are assigned the `web` middleware group, which provides features like session state and CSRF protection. The routes in `routes/api.php` are stateless and are assigned the `api` middleware group.

For most applications, you will begin by defining routes in your `routes/web.php` file.

```
Route::get($uri, $callback);  
Route::post($uri, $callback);  
Route::put($uri, $callback);  
Route::patch($uri, $callback);  
Route::delete($uri, $callback);  
Route::options($uri, $callback);
```

<https://laravel.com/docs/8.x/routing>

Controllers

Controllers can group related request handling logic into a single class. Controllers are stored in the `app/Http/Controllers` directory.

<https://laravel.com/docs/8.x/controllers>

Controllers (Resource)

Laravel resource routing assigns the typical "CRUD" routes to a controller with a single line of code. For example, you may wish to create a controller that handles all HTTP requests for "photos" stored by your application. Using the `make:controller` Artisan command, we can quickly create such a controller:

```
php artisan make:controller PhotoController
```

This command will generate a controller at `app/Http/Controllers/PhotoController.php`. The controller will contain a method for each of the available resource operations.

<https://laravel.com/docs/8.x/controllers#resource-controllers>

Controllers (Resource) cont...

Next, you may register a resourceful route to the controller:

```
Route::resource('photos', 'PhotoController');
```

This single route declaration creates multiple routes to handle a variety of actions on the resource. The generated controller will already have methods stubbed for each of these actions, including notes informing you of the HTTP verbs and URIs they handle.

<https://laravel.com/docs/8.x/controllers#resource-controllers>

Controllers (Resource) cont...

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

<https://laravel.com/docs/8.x/controllers#resource-controllers>

Requests

To obtain an instance of the current HTTP request via dependency injection, you should type-hint the `Illuminate\Http\Request` class on your controller method.

```
namespace \App\Controllers;
use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Store a new user.
     *
     * @param Request $request
     * @return Response
     */
    public function store(Request $request)
    {
        $name = $request->input('name');

        //
    }
}
```

<https://laravel.com/docs/5.4/requests>

`request()`

The `request()` function returns the current `request` instance or obtains an input item:

```
$request = request();
$name = request('name', $default = null)
```

<https://laravel.com/docs/8.x/requests>

Requests (File)

You may access uploaded files from a `Illuminate\Http\Request` instance using the `file` method or using dynamic properties. The `file` method returns an instance of the `Illuminate\Http\UploadedFile` class, which extends the PHP `SplFileInfo` class and provides a variety of methods for interacting with the file:

<https://laravel.com/docs/8.x/requests#files>

```
$file = $request->file('photo');  
  
$file = $request->photo;
```

Responses

All routes and controllers should return a response to be sent back to the user's browser. Laravel provides several different ways to return responses. The most basic response is simply returning a string from a route or controller.

```
Route::get('/', function () {  
    return 'Hello World';  
});
```

In addition to returning strings from your routes and controllers, you may also return arrays. The framework will automatically convert the array into a JSON response:

```
Route::get('/', function () {  
    return [1, 2, 3];  
});
```

<https://laravel.com/docs/8.x/responses>

Responses (Download)

The `download` method may be used to generate a response that forces the user's browser to download the file at the given path.

The `download` method accepts a file name as the second argument to the method, which will determine the file name that is seen by the user downloading the file.

Finally, you may pass an array of HTTP headers as the third argument to the method:

```
return response()->download($pathToFile);  
return response()->download($pathToFile, $name, $headers);
```

<https://laravel.com/docs/8.x/responses#file-downloads>

Responses (Display File)

The `file` method may be used to display a file, such as an image or PDF, directly in the user's browser instead of initiating a download. This method accepts the path to the file as its first argument and an array of headers as its second argument:

```
return response()->file($pathToFile);  
return response()->file($pathToFile, $headers);
```

<https://laravel.com/docs/8.x/responses#file-responses>

Responses (Redirect)

Redirect responses are instances of the `Illuminate\Http\RedirectResponse` class, and contain the proper headers needed to redirect the user to another URL. There are several ways to generate a `RedirectResponse` instance. The simplest method is to use the global `redirect` helper:

```
Route::get('dashboard', function () {  
    return redirect('home/dashboard');  
});
```

When you call the `redirect` helper with no parameters, an instance of `Illuminate\Routing\Redirector` is returned, allowing you to call any method on the `Redirector` instance. For example, to generate a `RedirectResponse` to a named route, you may use the `route` method:

```
return redirect()->route('login');
```

Sometimes you may wish to redirect the user to their previous location, such as when a submitted form is invalid. You may do so by using the global `back` helper function. Since this feature utilizes the `session`, make sure the route calling the `back` function is using the `web` middleware group or has all of the session middleware applied:

```
Route::post('user/profile', function () {  
    // Validate the request...  
    return back()->withInput();  
});
```

<https://laravel.com/docs/8.x/responses#redirects>

Views

Views contain the HTML served by your application and separate your controller / application logic from your presentation logic. Views are stored in the `resources/views` directory. A simple view might look something like this:

```
<!-- View stored in
resources/views/greeting.blade.php -->

<html>
  <body>
    <h1>Hello, {{ $name }}</h1>
  </body>
</html>
```

Since this view is stored at `resources/views/greeting.blade.php`, we may return it using the global `view` helper like so:

```
Route::get('/', function () {
    return view('greeting', ['name' =>
        'James']);
});
```

<https://laravel.com/docs/8.x/views>

Blade Templates

Blade is the simple, yet powerful templating engine provided with Laravel. Unlike other popular PHP templating engines, Blade does not restrict you from using plain PHP code in your views. In fact, all Blade views are compiled into plain PHP code and cached until they are modified, meaning Blade adds essentially zero overhead to your application. Blade view files use the `.blade.php` file extension and are typically stored in the `resources/views` directory.

In this lesson, you will learn most of the essential usage in Blade - definition, control structures, etc.

<https://laravel.com/docs/8.x/blade>

Blade Templates

```
Hello, {{ $name }}.
```

```
The current UNIX timestamp is {{ time() }}.
```

```
{{ $name or 'Default' }}
```

<https://laravel.com/docs/8.x/blade>

Blade Templates

Control Structure

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) >
1)
    I have multiple records!
@else
    I don't have any
records!
@endif

@unless (Auth::check())
    You are not signed in.
@endunless
```

Loops

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

```
@foreach ($users as $user)
    @if($user->type == 1)
        @continue
    @endif

    <li>{{ $user->name }}</li>

    @if($user->number == 5)
        @break
    @endif
@endforeach

@foreach ($users as $user)
    @continue($user->type == 1)

    <li>{{ $user->name }}</li>

    @break($user->number == 5)
@endforeach
```

<https://laravel.com/docs/5.4/blade#control-structures>

Blade Templates

Include Sub-Views

```
<div>
  @include('shared.errors')

  <form>
    <!-- Form Contents -->
  </form>
</div>

@include('view.name', ['some' => 'data'])
```

Stacks

```
@push('scripts')
  <script src="/example.js"></script>
@endpush
```

CSRF & Method Spoofing

CSRF	Method Spoofing
<ol style="list-style-type: none">1. Helps against Cross-site Request Forgery attacks2. Required to include in every form <pre data-bbox="459 543 697 568">{{ csrf_field() }}</pre>	<p data-bbox="1029 418 1750 495">HTML forms only allow the GET and POST HTTP verbs, so we need a way to spoof a DELETE request from the form. This is required for form other than GET and POST - PUT, PATCH, DELETE.</p> <pre data-bbox="1213 530 1586 555">{{ method_field('DELETE') }}</pre> <pre data-bbox="1058 587 1734 612"><input type="hidden" name="_method" value="DELETE"></pre>

<https://laravel.com/docs/5.4/csrf>

<https://laravel.com/docs/5.4/routing#form-method-spoofing>

[TIPS] Build Bootstrap form easily with Bootsnipp - <http://bootsnipp.com/forms>

How to Create a Page

1. Route

1. Open `routes/web.php`
2. Add your [route](#)

2. Controller

1. Create a controller that used in your route

```
php artisan make:controller YourController --resource
```
2. Open up `app/Http/Controllers/YourController.php`
3. Add relevant method as used in your route

```
public function index() {  
    return view('your.view');  
}
```

3. View

1. Create a folder named `your` in `resources/views`
2. Create `view.blade.php` in `resources/views/your` folder

Exercise

Create New Project and Pages

1. Include Basic Auth
2. Pages
 - a. About Us
 - b. Contact Us

Database

Migrations

Factory

Seeding

Migrations

“Migrations are like version control for your database”

```
php artisan make:migration create_tasks_table --create=tasks
```

```
php artisan make:migration update_tasks_table --table=tasks
```


Migrations

“Common column types used”

```
$table->increments('id');  
$table->integer('user_id');  
$table->string('password', 64);  
$table->enum('race', ['Malay', 'Indian', 'Chinese', 'Others']);  
$table->text('description');
```

Factory

“Just tell the factory how a default model should look.”

```
$factory->define(App\User::class, function (Faker\Generator $faker) {  
    return [  
        'name' => $faker->name,  
        'email' => $faker->email,  
        'password' => bcrypt(str_random(10)),  
        'remember_token' => str_random(10),  
    ];  
});
```

Factory (Create Dummy Data)

Run tinker

Create a user

Create 10 users

Seeding

“A simple method of seeding your database with test data using seed classes”

```
class CustomerSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::table('customers')->truncate();
        factory(App\Customer::class, 300)->create();
        $this->command->info('Customer table seeded!');
    }
}
```

<https://scotch.io/tutorials/generate-dummy-laravel-data-with-model-factories>

Exercise

Create 2 migration scripts

1. Create ``profiles`` table with following columns
 - a. `id`
 - b. `user_id`
 - c. `timestamps()`
2. Update ``profiles`` table by adding following columns
 - a. `religion - Islam, Christian, Hindu, Others`
 - b. `race - Malay, Indian, Chinese, Others`

Create a Profile Factory

1. Create profile factory based on previous migration scripts

Create a User Seeding

1. Create a ``users`` table seeder, ensured to create relationship data too.

Eloquent

Define

Queries

Relationships

Query Relations

Collections

Eloquent

Create a `app/Post.php` model

```
php artisan make:model Post
```

Create a `app/Models/Post.php` model

```
php artisan make:model Models/Post
```

Create a `app/Models/Post.php` model + migration script

```
php artisan make:model Models/Post -m
```

Eloquent

```
$flight = App\Flight::find(1);
```

Find a flight with id = 1

```
$flight = App\Flight::where('active', 1)->first();
```

Find the first active flight

```
$model = App\Flight::findOrFail(1);
```

Find a flight with id = 1 or throw an exception

```
$model = App\Flight::where('legs', '>', 100)->findOrFail();
```

Find the first flight with more than 100 legs or throw an exception

```
$count = App\Flight::where('active', 1)->count();
```

Count active flights

```
$max = App\Flight::where('active', 1)->max('price');
```

Get most expensive price for active flights

Eloquent

Create A Resource

```
$flight = new Flight;
$flight->name = $request->name;
$flight->save();

App\Flight::create(['name' => $request->name]);
```

Delete Resource

```
App\Flight::destroy(1);
App\Flight::where('id', 1)->delete();
App\Flight::destroy([1, 2, 3]);
App\Flight::destroy(1, 2, 3);
```

Update Resource

```
App\Flight::where('active', 1)
    ->where('destination', 'San Diego')
    ->update(['delayed' => 1]);

App\Flight::where(id, 1)
    ->update($request->only(['destination']));
```

Eloquent (Relationships)

One to One	One to Many
<p>A user has a profile.</p> <p>A user has a role.</p>	<p>A user have many photos.</p> <p>A user have many books.</p>
Many to One	Many to Many
<p>Many photos belongs to a user.</p> <p>Many books belongs to a user.</p>	<p>Many users have many photos.</p> <p>Many users have many books.</p>

Eloquent (Eager and Lazy)

Eager

```
$books = App\Book::with('author')->get();

// query book with author & publisher

$books = App\Book::with('author', 'publisher')->get();

// query book with author & author's contacts

$books = App\Book::with('author.contacts')->get();

// query user with post constraint

$users = App\User::with(['posts' => function ($query) {
    $query->where('title', 'like', '%first%');
}])->get();
```

Lazy

```
// query book with author

$books = App\Book::all();

if ($someCondition) {
    $books->load('author', 'publisher');
}

// query load author with constraint

$books->load(['author' => function ($query) {
    $query->orderBy('published_date', 'asc');
}]);
```

Eloquent (Collections)

All multi-result sets returned by Eloquent are instances of the `Illuminate\Database\Eloquent\Collection` object, including results retrieved via the `get` method or accessed via a relationship.

The Eloquent collection object extends the Laravel [base collection](#), so it naturally inherits dozens of methods used to fluently work with the underlying array of Eloquent models.

<https://laravel.com/docs/8.x/eloquent-collections>

Eloquent (Collections) cont.

<https://laravel.com/docs/8.x/eloquent-collections#available-methods>

all	isNotEmpty	shuffle
average	keyBy	slice
avg	keys	sort
chunk	last	sortBy
collapse	map	sortByDesc
combine	mapWithKeys	splice
contains	max	split
containsStrict	median	sum
count	merge	take
diff	min	tap
diffKeys	mode	toArray
each	nth	toJson
every	only	transform
except	partition	union
filter	pipe	unique
first	pluck	uniqueStrict
flatMap	pop	values
flatten	prepend	when
flip	pull	where
forget	push	whereInStrict
forPage	put	whereIn
get	random	whereInStrict
groupBy	reduce	whereNotIn
has	reject	whereNotInStrict
implode	reverse	zip
intersect	search	
isEmpty	shift	

Exercise

Create Users Management Pages

1. Should have all basic pages for create, read, update and delete.
2. Having relationships for at least one relationship type.

Exercise

Create a mind map of what you have learnt so far. You can work as a team. You have 30 minutes.

List out all the artisan commands you learn so far and describe the usage each of it.

Security

Authentication - (covered in [Authentication - Quickstart](#))

Authorization - (covered in [Using Packages - Laratrust](#))

Validation

Validation Trait

Display Error Messages

Form Request Validations

Custom Error Messages

Available Rules

Multilingual Validation Messages

Validation (Validation Trait)

In Controller

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);
    // The blog post is valid, store in database...
}
```

Stop on First Validation Failure

```
$this->validate($request, [
    'title' => 'bail|required|unique:posts|max:255',
    'body' => 'required',
]);
```

<https://laravel.com/docs/5.4/validation#quick-writing-the-validation-logic>

Validation (Display Error Message)

```
<!-- /resources/views/post/create.blade.php -->

<h1>Create Post</h1>

@if ($errors->any())
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

<!-- Create Post Form -->
```

<https://laravel.com/docs/5.4/validation#quick-displaying-the-validation-errors>

Validation (Form Request Validation)

Create a Request

```
php artisan make:request StorePostRequest
```

Open app/Http/Requests/StorePostRequest.php

```
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

Type-hint the Request

```
public function store(StoreBlogPostRequest $request)
{
    // The incoming request is valid...
    // Don't forget to include StorageBlogPostRequest
    // namespace at the top
}
```

<https://laravel.com/docs/5.4/validation#form-request-validation>

Validation (Custom Error Message)

```
/**
 * Get the error messages for the defined validation rules.
 *
 * @return array
 */
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'body.required' => 'A message is required',
    ];
}
```

Validation (Rules)

Accepted	Dimensions (Image Files)	Numeric
Active URL	Distinct	Present
After (Date)	E-Mail	Regular Expression
After Or Equal (Date)	Exists (Database)	Required
Alpha	File	Required If
Alpha Dash	Filled	Required Unless
Alpha Numeric	Image (File)	Required With
Array	In	Required With All
Before (Date)	In Array	Required Without
Before Or Equal (Date)	Integer	Required Without All
Between	IP Address	Same
Boolean	JSON	Size
Confirmed	Max	String
Date	MIME Types	Timezone
Date Format	MIME Type By File Extension	Unique (Database)
Different	Min	URL
Digits	Nullable	
Digits Between	Not In	

<https://laravel.com/docs/5.4/validation#available-validation-rules>

Validation (Multilingual Validation Message)

1. Make sure to Configure your [Localization](#)
2. Duplicate **resources/lang/en** to **resources/lang/xx**
3. Update the **resources/lang/xx/validation.php** based on your language setup.

Others

Using Packages

Helpers

Using Packages (Laravel Debugbar)

<https://github.com/barryvdh/laravel-debugbar>

Using Packages (Flash)

<https://github.com/laracasts/flash>

Using Packages (Former)

<https://github.com/formers/former>

Using Packages (Laratrust)

<https://github.com/santigarcor/laratrust>

Helpers

Laravel includes a variety of global "helper" PHP functions. Many of these functions are used by the framework itself; however, you are free to use them in your own applications if you find them convenient.

Laravel helpers categorised by:

1. Arrays
2. Paths
3. Strings
4. URLs
5. Miscellaneous

<https://laravel.com/docs/5.4/helpers>

Project

Requirements

1. Draw a mindmap on a single paper, on what you understand about Laravel Framework
2. Create a Laravel project with:
 - a. Authentication
 - b. Authorization
 - i. Administrator
 - ii. Staff
 - c. Permissions
 - i. CRUD Users
 - ii. CRUD Tasks
 - d. User Manager
 - i. Only Administrator can CRUD users
 - e. Task Manager
 - i. Administrator can CRUD tasks
 - ii. Staff can read & update only

Additional Readings

1. Working With IIS8 - <https://laracasts.com/discuss/channels/servers/iis-8-laravel-webconfig>