

v-for

```
<ul>
  <li v-for="value in users">{{ value }}</li>
</ul>
</div>

<script src="https://unpkg.com/vue@2.1.10/dist/vue.js"></script>
<script type="text/javascript">

var app = new Vue({
  el : '#app',
  data : {
    users : ["hesam" , "reza" , "mohammad" , "ali"]
  }
});
```

- hesam
- reza
- mohammad
- ali

index

```
<div id="app">
  <!-- Code -->
  <ul>
    <li v-for="(value , index) in users">{{index}} - {{ value }}</li>
  </ul>
</div>
```

- 0 - hesam
- 1 - reza
- 2 - mohammad
- 3 - ali

pass data to it v-bind or :

```
data : {
  articles : [
    {
      title : "Project Name 1",
      image : "http://roocket.ir/public/image/2017/2/13/interesting-libraries-february-2017.png"
      body : "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam viverra euismod odio, gravida pellentesque urna"
    },
    {
      title : "Project Name 1",
      image : "http://roocket.ir/public/image/2017/2/13/interesting-libraries-february-2017.png"
      body : "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam viverra euismod odio, gravida pellentesque urna"
    },
    {
      title : "Project Name 1",
      image : "http://roocket.ir/public/image/2017/2/13/interesting-libraries-february-2017.png"
      body : "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam viverra euismod odio, gravida pellentesque urna"
    }
  ]
}
```

```

```

v-on or @

به یه چیزی گوش میدیم تا یه چیزی بگیریم از ورودی we listen to sth to receive sth from our template

و خوب به یه ایونت گوش میکنه، چون اینجا تو یه button تعریف شده پس به طور پیش فرض از هر dom event که برای button هست مثل click, mouse enter, mouse leave,... میتونه استفاده کنه.

```
<div id="app" class="container">
  <!-- Code -->
  <div class="row">
    <div class="col-lg-12">
      <h1>است {{ counter }} تعداد کلیک های شما</h1>
      <button type="button" id="plus" v-on:click="counter++">کلیک</button>
    </div>
  </div>
</div>
```

```
<input type="text" v-on:keyup.enter.space="alertMe">
```

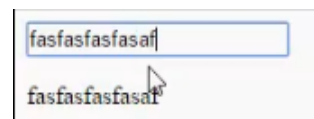
Vue provides aliases for the most commonly used key codes when necessary for legacy browser support:

- .enter
- .tab
- .delete (captures both "Delete" and "Backspace" keys)
- .esc
- .space
- .up
- .down
- .left
- .right

در v-model ترکیب این دوتا بالایی یعنی : و @ هست و دیتا دوطرفه جریان داره

```
<div id="app">
  <input type="text" v-model="message">
  <p>{{ message }}</p>
</div>

<script src="https://unpkg.com/vue@2.1.10/dist/vue.js"></script>
<script type="text/javascript">
  new Vue({
    el : '#app',
    data : {
      message : "Hello Vue"
    }
  })
</script>
```



v-once

در اینجا title فقط یکبار render میشود. مقدار اولیه را نشان میدهد نه وقتی که

```
<h1 v-once>{{ title }}</h1>
```

عوض میشود را.

v-html

finishedLink را به عنوان کد html رندر میکند نه تکست.

```
<p v-html="finishedLink"></p>
```

Event modifiers

Event Modifiers

It is a very common need to call `event.preventDefault()` or `event.stopPropagation()` inside event handlers. Although we can do this easily inside methods, it would be better if the methods can be purely about data logic rather than having to deal with DOM event details.

To address this problem, Vue provides event modifiers for `v-on`. Recall that modifiers are directive postfixes denoted by a dot.

- `.stop`
- `.prevent`
- `.capture`
- `.self`
- `.once`

```
<p v-on:mousemove="updateCoordinates">
  Coordinates: {{ x }} / {{ y }}
  - <span v-on:mousemove.stop="">DEAD SPOT</span>
</p>
```

Method متدها با هر تغییری run میشوند

```
<ul class="list-group">
  <li class="list-group-item" v-for="(todo , index) in todos"><a href="#" v-on:click="removeTodo(index)">{{ todo }}</a></li>
</ul>
<input type="text" class="form-control" v-model="newTodo" placeholder="کُز جدید" v-on:keyup.enter="addTodo">
```

```
<script type="text/javascript">

let app = new Vue({
  el : '#app',
  data : {
    newTodo : '',
    todos : [
      'انجام پروژه 1',
      'اقرار دادن پست در سایت',
      'بازی فوتبال'
    ]
  },
  methods : {
    addTodo() {
      if(this.newTodo !== '') {
        this.todos.push(this.newTodo);
        this.newTodo = '';
      }
    },
    removeTodo(index) {
      this.todos.splice(index,1);
    }
  }
});

</script>
```

```

```

کلاس image در صورتی به img اضافه میشه که activeClass مقدارش true باشه.

اطلاعات بیشتر در سایت ویو در قسمت style and class binding

Computed به صورت سنکرون عمل میکند و اگر آرگیومننت مربوط به آن تغییر کند عمل میکنند و مثل data میتوان به آن دسترسی پیدا کرد.

```
computed : {
  completeTodos() {
    return this.todos.filter(todo => todo.complete);
  }
}
```

Watch به صورت آسنکرون است

```
watch: {
  counter: function(value) {
    var vm = this;
    setTimeout(function() {
      vm.counter = 0;
    }, 2000);
  }
}
```

قانون نوشتن کامپوننت ها

```
<my-button></my-button>
```

```
Vue.component('myButton' , {  
  template : '<div>Hello Vue</div>'  
});
```

ایجاد کامپوننتها

```
<body>  
  <div id="app" class="container">  
    <!-- Code -->  
    <div class="row">  
      <div class="col-lg-12">  
        <my-button></my-button>  
        <my-button></my-button>  
        <my-button></my-button>  
        <my-button></my-button>  
      </div>  
    </div>  
  </div>  
</body>  
  
<script src="https://unpkg.com/vue@2.1.10/dist/vue.js"></script>  
<script type="text/javascript">  
  
  Vue.component('myButton' , {  
    template : '<button v-on:click="counter++">{{ counter }}</button>',  
    data() {  
      return {  
        counter : 0  
      }  
    }  
  });  
  
  let app = new Vue({  
    el : '#app',  
  });  
  
</script>  
</body>
```

```

<div id="app" class="container">
  <!-- Code -->
  <div class="row">
    <div class="col-lg-12">
      <div style="margin-top:50px"></div>
      <alert title="اخطار" message="موفقیت انجام نشد" type="warning"></alert>
      <alert title="موفقیت" message="موفقیت انجام شد" type="success"></alert>
    </div>
  </div>
</div>

<script src="https://unpkg.com/vue@2.1.10/dist/vue.js"></script>
<script type="text/javascript">

Vue.component('alert' , {
  props : ['title' , 'message' , 'type'],
  template : `
    <div :class="['alert' , classAlert ]" role="alert">
      <button type="button" class="close">
        <span aria-hidden="true">x</span>
      </button>
      <strong>{{ title }}!</strong> {{ message }}.
    </div>
  ` ,
  data() {
    return {
      classAlert : `alert-${this.type}`
    }
  }
});

```

v-show

```

Vue.component('alert' , {
  props : ['title' , 'message' , 'type'],
  template : `
    <div :class="['alert' , classAlert ]" role="alert" v-show="isActive">
      <button type="button" class="close" @click="isActive = false">
        <span aria-hidden="true">x</span>
      </button>
      <strong>{{ title }}!</strong> {{ message }}.
    </div>
  ` ,
  data() {
    return {
      classAlert : `alert-${this.type}`,
      isActive : true
    }
  }
});

```

میشه به جای این از v-if استفاده کرد. تفاوتشون اینه که v-show میاد display رو none میکنه، v-if کلا پاکش میکنه

```
<p v-if="show">You can see me!</p>
<p v-else>Now you see me!</p>
```

علاوه بر v-else-if، v-else هم داریم.

اگر بخواهیم v-if به چند المان اعمال شود و نخواهیم درون div بگذاریمشان، در template میذاریم

```
<template v-if="show">
  <h1>Heading</h1>
  <p>Inside a template</p>
</template>
```

Refs میتونی رو هر المان html که بخوای بذاریش

```
<button v-on:click="show" ref="myButton">Show
Paragraph</button>
```

داخل کد html

داخل اسکریپت `vm1.$refs.heading.innerText = 'Something else';` و با استفاده از ویو (vm1) الان المان ویو

هست (میتونی بهش با اسمش دسترسی داشته باشی)

چگونه از یک کامپوننت به کامپوننت دیگر مقدار میفرستیم؟ از طریق المان پدر، به این ترتیب

```
export default {
  props: ['userAge'],
  methods: {
    editAge() {
      this.age = 30;
    }
  }
}
```

userAge در هر دو کامپوننت تعریف میشود

```
props: ['userAge'],
methods: {
  editAge() {
    this.userAge = 30;
    this.$emit('ageWasEdited', this.userAge);
  }
}
```

با استفاده از \$Emit به المان پدر مقدار userAge را

میفرستیم.

در المان پدر یک listener تعریف میکنیم که متوجه تغییر userAge میشود.

```
export default {
  data: function () {
    return {
      name: 'Max',
      age: 27
    };
  },
  methods: {
```

```
<div class="col-xs-12 col-sm-6">
  <app-user-edit
    :userAge="age"
    @ageWasEdited="age = $event"
  ></app-user-edit>
</div>
```

در این حالت چون age به عنوان prop به کامپوننت دیگر پاس داده شده در صورت تغییر در المان اول در المان دوم هم عوض میشود.

راه حل دیگر: در main.js یا app.js این را تعریف میکنیم

این را باید زودتر از کامپوننتای دیگه تعریف کنیم.

```
export const EventBus = new Vue();
```


و در تمام کامپوننت ها export میکنیم
اینجوری:

```
import { EventBus } from '../main';

export default {
  props: ['userAge'],
  methods: {
    editAge() {
      this.userAge = 30;
      this.$emit('ageWasEdited', this.userAge);
      → EventBus.$emit('ageWasEdited', this.userAge);
    }
  }
}
```

در کامپوننتی که میخوایم

عوضش کنیم این شکلی میشه

تو اون یکی کامپوننت که قراره با تغییر این عوض بشه به listener تعریف میکنیم در created()

```
created() {
  EventBus.$on('ageWasEdited', (age) => {
    this.userAge = age;
  });
}
```

تفاوت بین اینها اینه که بین هر دو کامپوننتی که بخوای میشه ازش استفاده کرد و لازم نیست به کامپوننت پدر بفرستن چیزا رو،
ميفرستن به EventBus

راه حل جامعتر این که در main.js اینگونه تعریف کنیم

```
export const EventBus = new Vue({
  methods: {
    changeAge(age) {
      this.$emit('ageWasEdited', age);
    }
  }
});
```

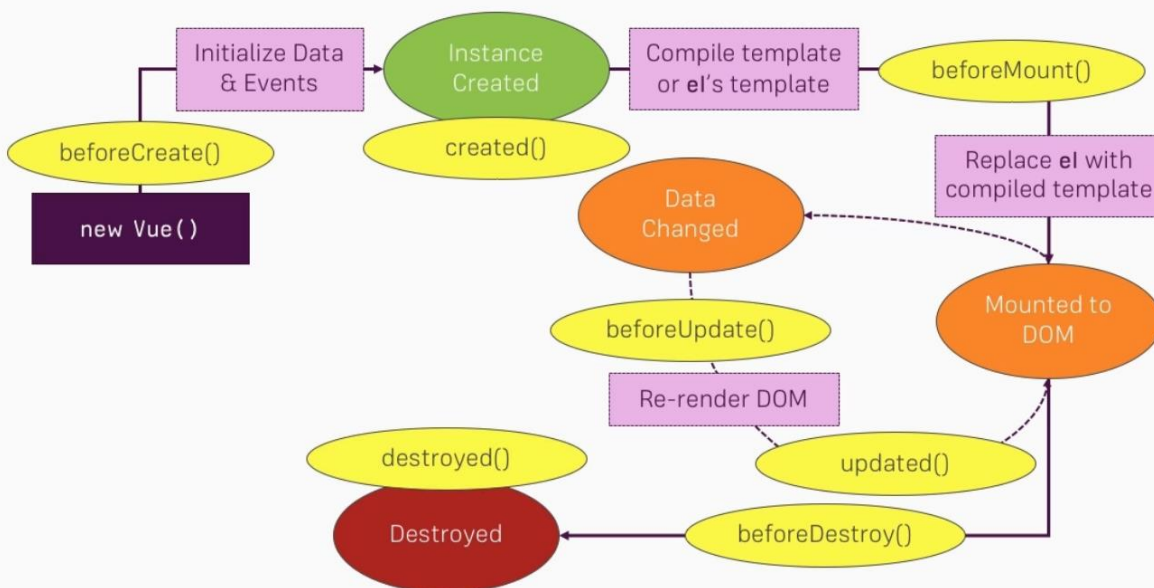
و در بقیه کامپوننت ها اینگونه استفاده کنیم

```
export default {
  props: ['userAge'],
  methods: {
    editAge() {
      this.userAge = 30;
      this.$emit('ageWasEdited', this.userAge);
      EventBus.$emit('ageWasEdited', this.userAge);
      → EventBus.changeAge(this.userAge);
    }
  }
}
```

10...

Life cycle

VueJS Instance Lifecycle



How to use data and methods in component

```
Vue.component('my-cmp', {
  data: function() {
    return {
      status: 'Critical'
    },

```

```
<script>
export default {
  data: function() {
    return {
      status: 'Critical'
    },
  },
  methods: {
    changeStatus() {
      this.status = 'Normal';
    }
  }
}
</script>
```

Component style اگر نخواهیم استایلی که به کامپوننت می‌دهیم به همه داده شود scoped را به تگ style اضافه می‌کنیم، اینجوری خودش یه کلاس می‌سازه به یه اسمی و اونو به المان‌های فقط همون کامپوننت می‌ده

```
<style scoped>
div {
  border: 1px solid red;
}
</style>
```

```
<jumbotron title="سلام دنیا!">
  نظر گرفته شده است (Lorem ipsum) : لورم ایپسوم یا طرح‌نما (به انگلیسی)
</jumbotron>
```

```
Vue.component('jumbotron', {
  props: ['title', 'message'],
  template: `
    <div class="jumbotron">
      <h1>{{ title }}</h1>
      <p><slot></slot></p>
      <p><a class="btn btn-primary btn-lg" href="#" role="button">مشاهده بیشتر</a></p>
    </div>
  `
})

new Vue({
  el: '#app'
})
```

```
Vue.component('jumbotron', {
  template: `
    <div class="jumbotron">
      <slot name="title"></slot>
      <p><slot></slot></p>
      <p><a class="btn btn-primary btn-lg" href="#" role="button">مشاهده بیشتر</a></p>
    </div>
  `
})

new Vue({
  el: '#app'
})
```

```
<jumbotron>
  <h1 slot="title">سلام دنیا!</h1>
  نه در نظر گرفته شده است (Lorem ipsum) : لورم ایپسوم یا طرح‌نما (به انگلیسی)
</jumbotron>
<!-- <jumbotron title="سلام دنیا!" message="لورم ایپسوم یا طرح‌نما (به انگلیسی)">
```

میتواند در صورتی که بهش متنی اختصاص ندیم خودش به صورت پیش فرض یه متن داشته باشه

```
<div class="title">
  <slot name="title"></slot>
  <span style="color: #ccc"><slot name="subtitle">The Subtitle</slot></span>
</div>
```

Dynamic component

```
export default {
  data: function() {
    return {
      quoteTitle: 'The Quote',
      selectedComponent: 'appQuote'
    },
  },
  components: {
    appQuote: Quote,
    appAuthor: Author,
    appNew: New
  }
}
```

به صورت default اون کامپوننتی که گذاشتیم استفاده میشه.

```
<button @click="selectedComponent = 'appQuote'">Quote</button>
<button @click="selectedComponent = 'appAuthor'">Author</button>
<button @click="selectedComponent = 'appNew'">New</button>
```

با کلیک کردن روی هرکدام از این button ها کامپوننتی که بهش وصلن قراره نمایش داده بشه، برای بقیه لازمه تا کامپونها رو bind کنیم

Bind components

در یک کامپوننت کامپوننت دیگر را اینگونه bind میکنیم با کلمه کلیدی is

```
<component :is="selectedComponent"></component>
```

Component tag بهمون اجازه میده تا کامپوننت ها رو به صورت دینامیک بذاریم. در همین کامپوننتی که دستور بالا رو گذاشتیم کامپوننت دیگه اینجوری import شده:

```
<script>
import Quote from './components/Quote.vue';
import Author from './components/Author.vue';
import New from './components/New.vue';

export default {
  data: function() {
    return {
      quoteTitle: 'The Quote',
      selectedComponent: 'appQuote'
    },
  },
  components: {
    appQuote: Quote,
  }
}
```

کاری که اینجا کرده اینه که اون Component

tag بالا رو bind کرده با : با یه data به نام SelectedComponent که حالا این SelectedComponent در اصل یه کامپوننت دیگه است که import کردیم به نام appQuote. (دیفالت، چون طبق بالا با کلیک روی button ها عوض میشه). نکته

مهم اینه که اینجا کامپوننتها در هر دفعه که button آنها کلیک میشود، دوباره ساخته میشن یعنی قبلی destroyed میشه. این حالت پیش فرضه، اگه بخوایم میتونیم عوضش کنیم:

```
<keep-alive>
  <component :is="selectedComponent">
    <p>Default Content</p>
  </component>
</keep-alive>
```

از تگ keep-alive استفاده میکنیم.

```
deactivated() {
  console.log('Deactivated!');
},
activated() {
  console.log('Activated!');
}
```

بخشی از life cycle هستند که میتونیم در صورت لزوم تو این حالت

ازشون استفاده کنیم.

Directive

میتوان اینگونه دایرکتیوها رو تعریف کرد. دایرکتیو اینجا میشه v-highlight مثل v-for

```
Vue.directive('highlight');
```

مثلا. اگه بخوایم global ازش استفاده کنیم، اینو تو main.js مینویسیم.

Directive lifecycle

Hooks

bind(el, binding, vnode)	Once Directive is Attached
inserted(el, binding, vnode)	Inserted in Parent Node
update(el, binding, vnode, oldVnode)	Once Component is Updated (without Children)
componentUpdated(el, binding, vnode, oldVnode)	Once Component is Updated (with Children)
unbind(el, binding, vnode)	Once Directive is Removed

```
Vue.directive('highlight', {
  bind(el, binding, vnode) {
    // el.style.backgroundColor = 'green';
    el.style.backgroundColor = binding.value;
  }
});
```

bind کردن

هرچی اینجا بین "" میذاریم به عنوان value در نظر گرفته میشود `<p v-highlight="'red'">Color this</p>`

```
Vue.directive('highlight', {
  bind(el, binding, vnode) {
    // el.style.backgroundColor = 'green';
    // el.style.backgroundColor = binding.value;
    if (binding.arg == 'background') {
      el.style.backgroundColor = binding.value;
    }
  }
});
```

حالت دیگر این است که

آرگومننت تعریف کنیم.

`<p v-highlight:background="'red'">Color this</p>`

```
Vue.directive('highlight', {
  bind(el, binding, vnode) {
    // el.style.backgroundColor = 'green';
    // el.style.backgroundColor = binding.value;
    var delay = 0;
    if (binding.modifiers['delayed']) {
      delay = 3000;
    }
    setTimeout(() => {
      if (binding.arg == 'background') {
        el.style.backgroundColor = binding.value;
      } else {
        el.style.color = binding.value;
      }
    }, delay);
  }
});
```

حالت دیگر این است که از modifier ها

که به صورت آرایه هستند استفاده کنیم (مودیفایرها با . به یکدیگر وصل میشوند) همینجوری که نشون داده شده.

`<p v-highlight:background.delayed="'red'">Color this</p>`

اگر local directive میخوایم، تو اسکریپت اینجوری نشون میدیم.

```
<script>
  export default {
    directives: {
      'local-highlight': {
        bind(el, binding, vnode) {
          // ...
        }
      }
    }
  }
</script>
```

فیلتر

```

<template>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-sm-8 col-sm-offse
        <h1>Filters & Mixins</h1>
        <p>{{ text | toUppercase }}</p>
      </div>
    </div>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        text: 'Hello there!'
      },
      filters: {
        toUppercase(value) {
          return value.toUpperCase();
        }
      }
    }
  }
</script>

```

برای اعمال فیلتر از | (pipe) استفاده میشه، یه فیلتر

همیشه یه value میگیره و همیشه یه چیزی مثل همون value برمیگردونه (return)، این فیلتر local است

اگه بخوایم چندتا فیلتر اعمال کنیم همین `<p>{{ text | toUppercase | to-lowercase }}</p>`

طوری با | هی پشت سر هم میذاریمشون.

```

Vue.filter('to-lowercase', function(value) {
  return value.toLowerCase();
});

```

تعریف به صورت global

Mixins

```

export default {
  mixins: [fruitMixin],
  data() {
    return {
      text: 'Hello there!'
    }
  },
}

```

Global mixin

```

1 import Vue from 'vue'
2 import App from './App.vue'
3
4 Vue.filter('to-lowercase', function(value) {
5   return value.toLowerCase();
6 });
7
8 Vue.mixin({
9   created() {
10    console.log('Global Mixin - Created Hook');
11  }
12 });
13
14 new Vue({
15   el: '#app',
16   render: h => h(App)
17 })

```



```

export const fruitMixin = {
  data() {
    return {
      fruits: ['Apple', 'Banana', 'Mango', 'Melon'],
      filterText: ''
    }
  },
  computed: {
    filteredFruits() {
      return this.fruits.filter((element) => {
        return element.match(this.filterText);
      });
    }
  }
};

```

Transitions

```

<transition name="fade">
  <div class="alert alert-info" v-if="show">This is some Info</div>
</transition>

```

برای

اضافه کردن Transition به یک المان، آن را بین تگ Transition قرار می دهیم.

```

<style>
  .fade-enter {
    opacity: 0;
  }

  .fade-enter-active {
    transition: opacity 1s;
  }

  .fade-leave {
    /*opacity: 1;*/
  }

  .fade-leave-active {
    transition: opacity 1s;
    opacity: 0;
  }
</style>

```

برای نام گذاری کلاسها در css، به name ای که برای Transition در نظر

گرفتیم (اینجا fade) توجه میکنیم. 4 تا حالت اصلی داریم: enter, enter-active, leave, leave-active

```

<transition name="slide" type="animation">
  <div class="alert alert-info" v-if="show">This is some Info</div>
</transition>

```

برای type دو

حالت داریم که مشخص میکند با کدام یک از اینها Transition تمام شود: animation, transition

```

<transition name="fade" appear>
  <div class="alert alert-info" v-if="show">This is some Info</div>
</transition>

```

با گذاشتن

appear با لود شدن صفحه، اگر این المان در حال نمایش داده شدن باشد، از Transition ای که برایش تعریف کردیم استفاده میکند، اگر appear را ننویسیم استفاده نمیکند.


```

<transition
  enter-active-class="animated bounce"
  leave-active-class="animated shake"
>
  <div class="alert alert-info" v-if="show">This is some Info</div>
</transition>

```

اگر بخوایم

کلاس های css رو از کتابخانه css بگیریم باید کلاس ها رو مثل بالا override کنیم.

```

<transition :name="alertAnimation" mode="out-in">
  <div class="alert alert-info" v-if="show" key="info">This is some Info</div>
  <div class="alert alert-warning" v-else key="warning">This is some Warning</div>
</transition>

```

از key برای وقتی استفاده میکنیم که داریم کلاس های مختلفی به المان های transition میدهم تا کلاس مختصشان را به آنها دهد. در transition فقط در آن واحد میتوان یک المان نمایش داد برای همین از v-if, v-else استفاده میکنیم و v-show در این حالت کار نمیکند. وقتی از mode="out-in" استفاده میکنیم یعنی اول المانی داریم نشونش میدیم کامل بره، بعد المانی که میخوایم نشونش بدیم بیاد.

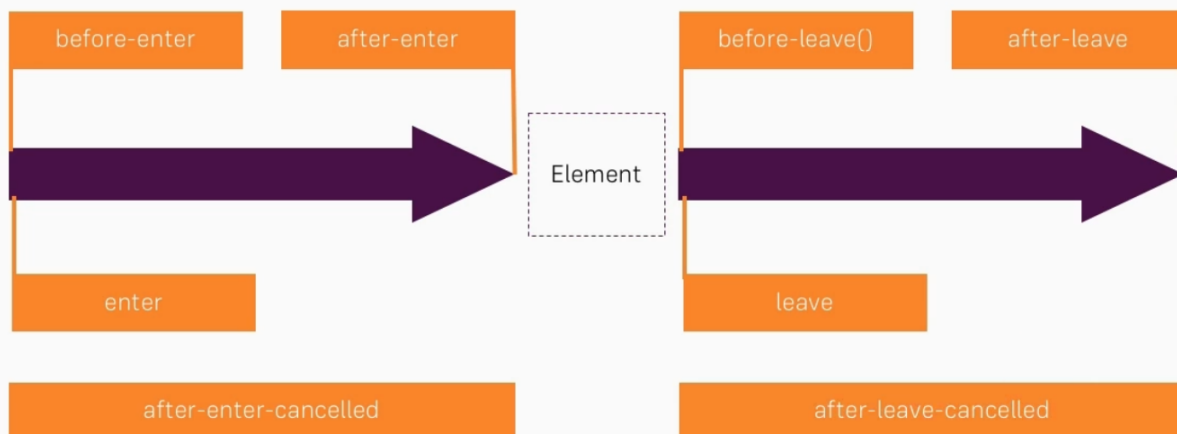
```

@keyframes flip-out {
  from {
    transform: rotateY(0deg);
  }
  to {
    transform: rotateY(90deg);
  }
}

```

وقتی خودمون کلاس مینویسیم با استفاده از CSS.

Transition JS Hooks



```

<transition
  @before-enter="beforeEnter"
  @enter="enter"
  @after-enter="afterEnter"
  @enter-cancelled="enterCancelled"

  @before-leave="beforeLeave"
  @leave="leave"
  @after-leave="afterLeave"
  @leave-cancelled="leaveCancelled">
  <div style="width: 100px; height: 100px; background-color: lightgreen" v-if="load">
</transition>

```

اگر نخواهیم از کلاس های CSS استفاده کنیم و مستقیم بریم سراغ JS این رو مینویسیم. `:css="false">`

```

methods: {
  beforeEnter(el) {
    console.log('beforeEnter');
  },
  enter(el, done) {
    console.log('enter');
    done();
  },
  afterEnter(el) {
    console.log('afterEnter');
  },
  enterCancelled(el) {
    console.log('enterCancelled');
  },
  beforeLeave(el) {
    console.log('beforeLeave');
  },
  leave(el, done) {
    console.log('leave');
    done();
  },
  afterLeave(el) {
    console.log('afterLeave');
  },
  leaveCancelled(el) {
    console.log('leaveCancelled');
  }
}

```

در enter, leave باید done رو فراخونی کنیم.

برای انیمیشن و ترنزیشن گروهی

```

<ul class="list-group">
  <transition-group name="slide">
    <li
      class="list-group-item"
      v-for="(number, index) in numbers"
      @click="removeItem(index)"
      style="cursor: pointer"
      :key="number">{{ number }}
    </li>
  </transition-group>
</ul>

```

باید از key حتما استفاده کنیم اینجا

تا بتونه تشخیص بده رو کدوم المان داره چیکار میکنه.

علاوه بر کلاس های قبلی که داشت (enter, enter-active, leave, leave-active) به transition-group یک کلاس پیش فرض دیگه هم اضافه شده : move

```
.slide-move {  
  transition: transform 1s;  
}
```

چرا اضافه شده؟ چون با حذف و اضافه شدن به المان به این لیست بقیه اعضا هم تکتون میخورن و لازمه که این هندل شه. حالا با این دستور اضافه شدن کامل هندل میشه ولی حذف شدن نه، چون هنوز المان قبلی جاش رو حفظ کرده پس برای اون لازمه این کارو بکنیم:

Position: absolute;

```
.slide-leave-active {  
  animation: slide-out 1s ease-out forwards;  
  transition: opacity 1s;  
  opacity: 0;  
  position: absolute; ←  
}
```

One important Difference:

<transition> is not rendered to the DOM!

<transition-group> does render a new HTML Tag!

By Default, that will be a , you can overwrite this by setting <transition-group tag="TAG">.

Vue with laravel

برای راه اندازی روی سرور از لاراوِل استفاده میکنیم توی کامند لاین این دستور رو میزنیم

```
composer create-project --prefer-dist laravel/laravel blog
```

به جای blog اسم پروژه رو میزنیم

بعد وارد فولدر پروژه میشیم و این دستور رو میزنیم

```
C:\Users\Roocket\Desktop\Project>npm i
```

که dependency هاشو نصب کنیم

```
C:\Users\Roocket\Desktop\Project>php artisan serve  
Laravel development server started: <http://127.0.0.1:8000>
```

با این دستور هم سرور راه میندازیم

```
C:\Users\Roocket\Desktop\Project>npm run watch
```

وقتی از vue در لاراوِل استفاده میکنیم برای این که با دستوری لاراوِل قاطی نشه قبلش از @ استفاده میکنیم

```

<body>
  <div id="app">
    <h1>@{{ msg }}</h1>
    <example></example>
  </div>

  <script src="/js/app.js"></script>
</body>

```

Vue in laravel

در فایل package.json ویو را وارد میکنیم

```

},
"devDependencies": {
  "axios": "^0.15.2",
  "bootstrap-sass": "^3.3.7",
  "jquery": "^3.1.0",
  "laravel-mix": "^0.6.0",
  "lodash": "^4.16.2",
  "vue": "^2.0.1"
}

```

و سپس دستور npm install را در command prompt در فولدر مربوط به پروژه run میکنیم تا این دپندنسی ها نصب بشن

سپس در فایل bootstrap.js این اطلاعات رو میذاریم

```

import Vue from 'vue';
import Axios from 'axios';

window.Vue = Vue;
window.axios = Axios;

window.axios.defaults.headers.common = {
  'X-CSRF-TOKEN': window.Laravel.csrfToken,
  'X-Requested-With': 'XMLHttpRequest'
};

```

Ajax

برای درخواست های Ajax یک پکیج درست شده واسه ویو به نام vue-resource

برای نصب دستور npm i -save vue-resource را وارد میکنیم.

```

App.vue × index.html × main.js ×
1 import Vue from 'vue'
2 import VueResource from 'vue-resource';
3 import App from './App.vue'
4
5 Vue.use(VueResource);
6

```

```

submit() {
  this.$http.post('https://vuejs-http.firebaseio.com/data.json', this.user)
    .then(response => {
      console.log(response);
    }, error => {
      console.log(error);
    });
}
متد

```

پست که مقدار اولی که میگیره url است و مقدار دوم چیزی که میفرستیم. حالا بعد از فرستادن، سرور یه چیزی برمیگردونه که ما اینجا با then میگیریمش که دو حالت داره: response و error که هرکاری بخوایم بکنیم باهاش اینجا میکنیم.

```

this.$http.get('https://vuejs-http.firebaseio.com/data.json')
  .then(response => {
    return response.json();
  })
  .then(data => console.log(data));

```

متد get، دوتا

then میذاریم چون کارش آسنکرونه و اگه then دوم رو نذاریم بهمون promise برمیگردونه نه data ای که میخوایم رو.

اگه بخوایم روت بذاریم:

```

App.vue × index.html × main.js ×
1 import Vue from 'vue'
2 import VueResource from 'vue-resource';
3 import App from './App.vue'
4
5 Vue.use(VueResource);
6
7 Vue.http.options.root = 'https://vuejs-http.firebaseio.com/data.json';

```

Interceptor

```

Vue.http.interceptors.push((request, next) => {
  console.log(request);
  if (request.method == 'POST') {
    request.method = 'PUT';
  }
  next(request);
});

```

برای هر request این

اجرا میشه و اینو تو main.js مینویسه.


```
Vue.http.interceptors.push((request, next) => {
  console.log(request);
  if (request.method == 'POST') {
    request.method = 'PUT';
  }
  next(response => {
    response.json = () => { return {messages: response.body} }
  });
});
```

اگه بخوایم واسه

response بنویسیم همیشه این شکلی!

Vue-resource default actions

Default Actions

```
get: {method: 'GET'},
save: {method: 'POST'},
query: {method: 'GET'},
update: {method: 'PUT'},
remove: {method: 'DELETE'},
delete: {method: 'DELETE'}
```

Resource

در فیلد دیتا آن را تعریف میکنیم. `resource: {}` سپس توی created این را میگوییم:

```
created() {
  this.resource = this.$resource('data.json');
}
```

که `$resource` از vue-resource میاد.

```
submit() {
  this.$http.post('data.json', this.user)
    .then(response => {
      console.log(response);
    }, error => {
      console.log(error);
    });
  this.resource.save({}, this.user);
}
```

که از default action استفاده میشه. اینجا تو متد

submit به جای دستور قبلی از این استفاده کردیم و به جواب هم گرفتیم. میتونیم action خودمون رو هم تعریف کنیم.

```
created() {
  const customActions = {
    saveAlt: {method: 'POST', url: 'alternative.json'}
  };
  this.resource = this.$resource('data.json', {}, customActions);
}
```

و اینطوری ازش استفاده کنیم:

```
methods: {
  submit() {
    this.$http.post('data.json', this.user)
      .then(response => {
        console.log(response);
      }, error => {
        console.log(error);
      });
    this.resource.save({}, this.user);
    this.resource.saveAlt(this.user);
  },
}
```

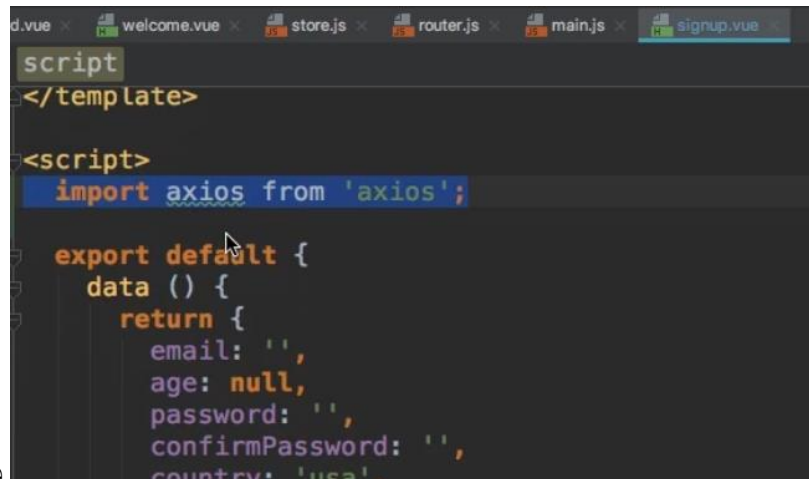
اینجوری `this.resource = this.$resource('{node}.json', {}, customActions);`

میتونیم دینامیک کنیم این جایی که میخوایم بهش مقدار بفرستیمو.

خب حالا میشه درخواست ها رو با axios هم فرستاد، پس:

Axios

npm i axios --save-dev



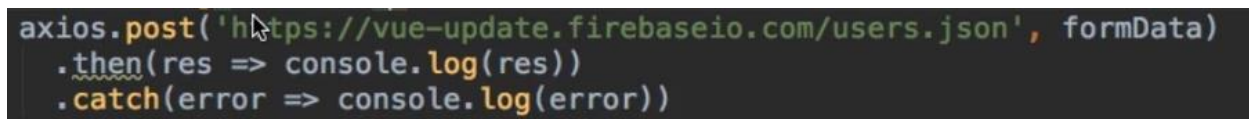
```
script
</template>

<script>
import axios from 'axios';

export default {
  data () {
    return {
      email: '',
      age: null,
      password: '',
      confirmPassword: '',
      country: 'usa'
    }
  }
}
```

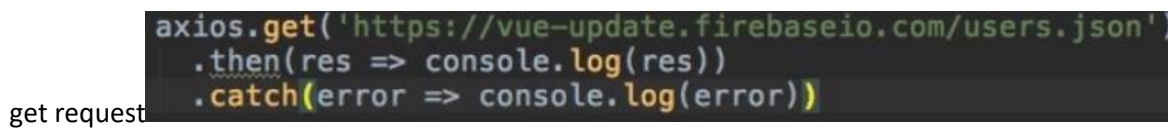
در هر کامپوننت میشه اینجوری import

کرد و ازش استفاده کرد.



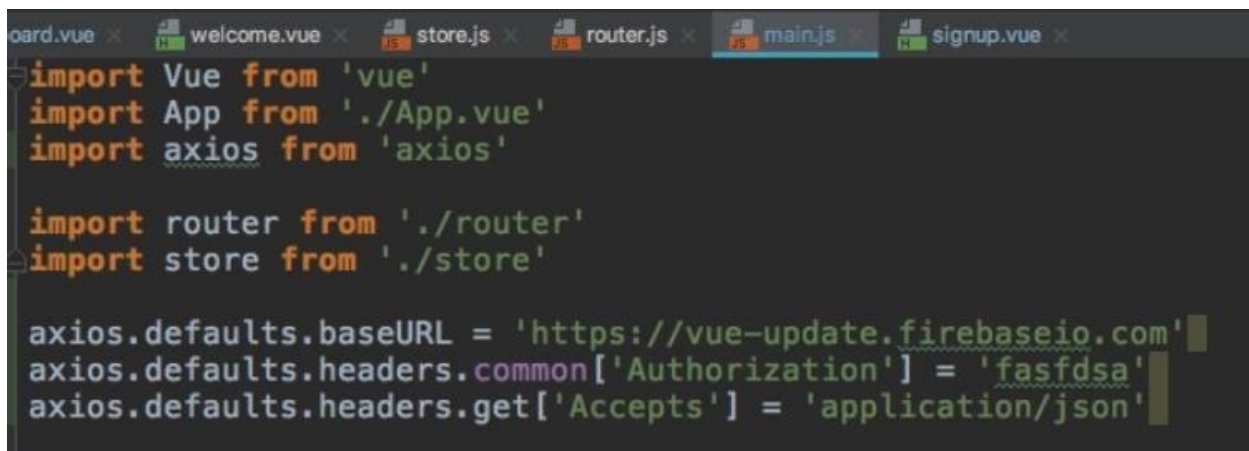
```
axios.post('https://vue-update.firebaseio.com/users.json', formData)
  .then(res => console.log(res))
  .catch(error => console.log(error))
```

post request, axios returns a promise



```
axios.get('https://vue-update.firebaseio.com/users.json')
  .then(res => console.log(res))
  .catch(error => console.log(error))
```

get request



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

import router from './router'
import store from './store'

axios.defaults.baseURL = 'https://vue-update.firebaseio.com'
axios.defaults.headers.common['Authorization'] = 'fasfdsa'
axios.defaults.headers.get['Accepts'] = 'application/json'
```

اگه یه آدرس دیفالت بخوایم بذاریم و قرار باشه بقیه آدرس ها relative این باشن.

Interceptor


```
const reqInterceptor = axios.interceptors.request.use(config => {
  console.log('Request Interceptor', config)
  return config
})
const resInterceptor = axios.interceptors.response.use(res => {
  console.log('Response Interceptor', res)
  return res
})
```

اینجوری تعریف

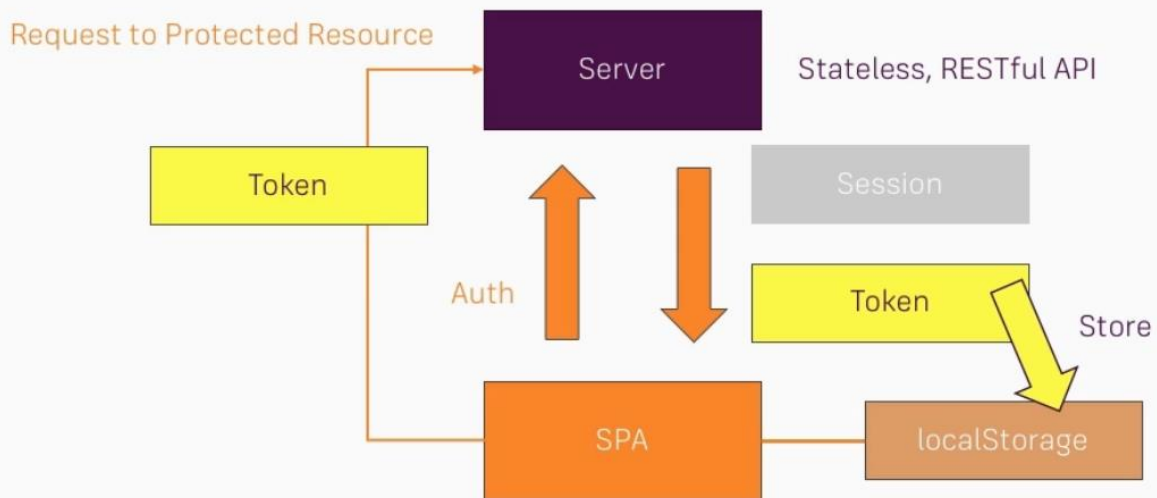
میکنیم در (main.js)

```
axios.interceptors.request.eject(reqInterceptor)
axios.interceptors.response.eject(resInterceptor)
```

و اینجوری پاکشون میکنیم.

Authentication

How Authentication Works in SPAs



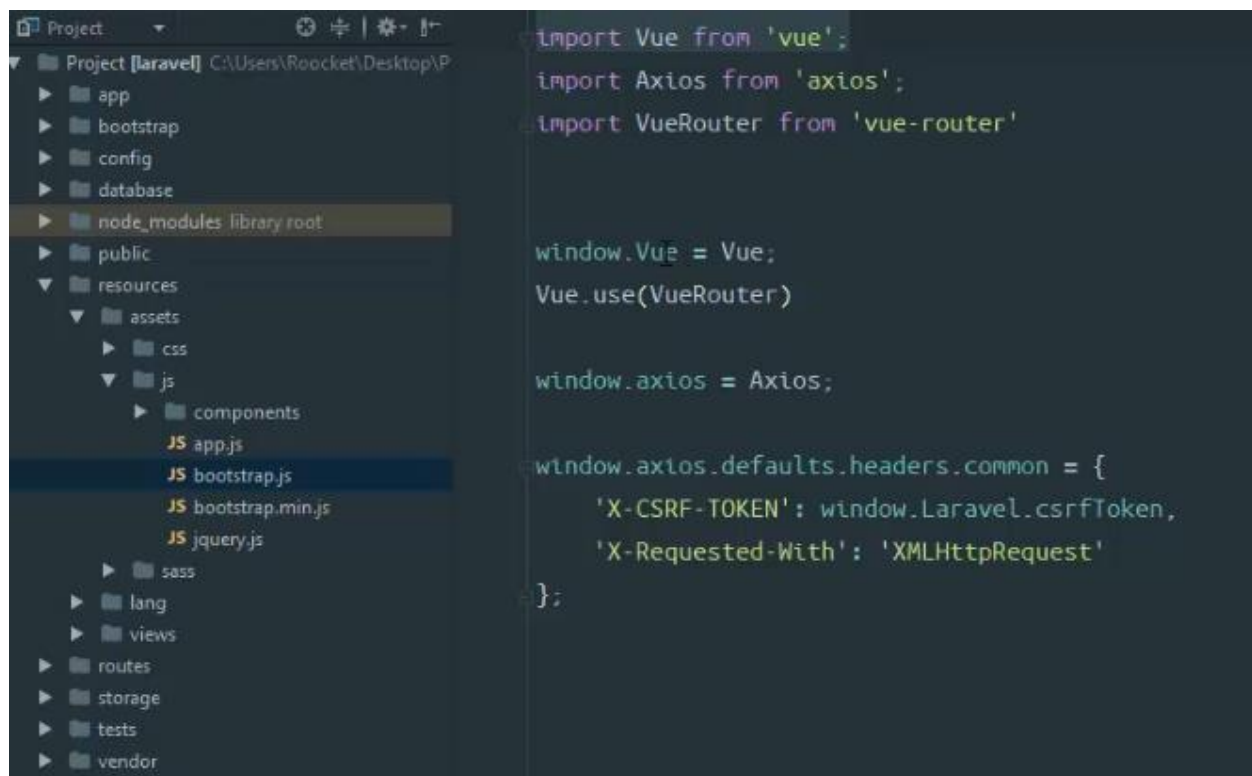
اینجا دیگه session نداریم و کار با token پیش میره. Token میتونه تبدیل به یه js object بشه، این آبجکت رو با هر ریکوئست که به سرور میفرستیم، میفرستیم.

16?!!

SPA

نصب ویو روتر

```
\Project>npm install vue-router --save-dev
```



در فولدر js یک فایل جدید درست میکنیم به نام router.js تا روت ها را آنجا تعریف کنیم



حال باید روت ها را اعمال کنیم

در blade مربوطه این کار را میکنیم

```

<div id="app">

  <my-header></my-header>
  <!-- Page Content -->
  <div class="container">

    <div class="row">

      <!-- Blog Entries Column -->
      <div class="col-md-8">

        <router-view></router-view>

      </div>

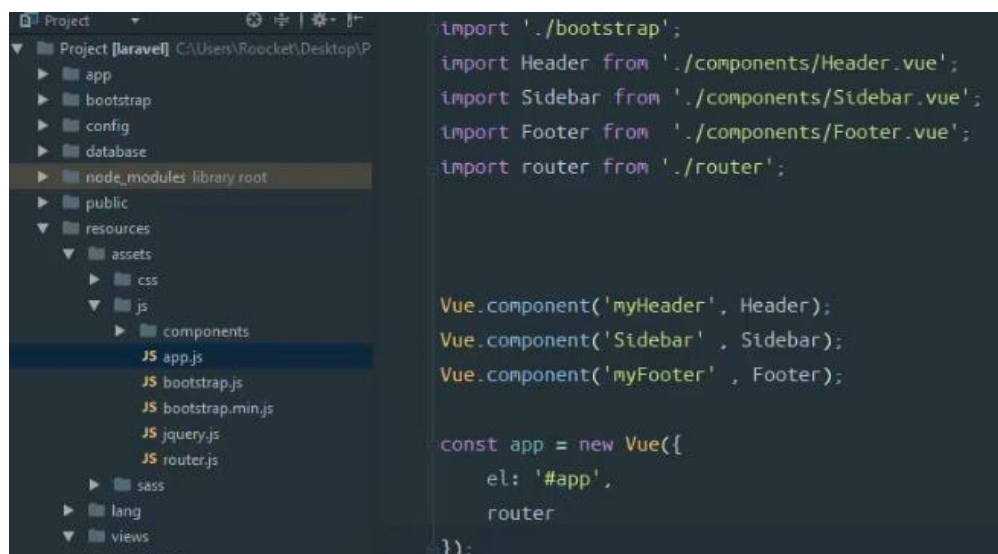
      <!-- Blog Sidebar Widgets Column -->
      <div class="col-md-4">
        <sidebar></sidebar>
      </div>

    </div>

  </div>
</div>

```

حال در app.js باید این کار را بکنیم



```

import './bootstrap';
import Header from './components/Header.vue';
import Sidebar from './components/Sidebar.vue';
import Footer from './components/Footer.vue';
import router from './router';

Vue.component('myHeader', Header);
Vue.component('Sidebar', Sidebar);
Vue.component('myFooter', Footer);

const app = new Vue({
  el: '#app',
  router
});

```

سپس دستور npm run watch را اجرا میکنیم

در آدرس این را خواهیم دید و این نشان دهنده این است که داریم از ویو استفاده میکنیم

```
Vue.use(VueRouter);

const router = new VueRouter({
  routes,
  mode: 'history'
});
```

no # mode

```
<template>
  <ul class="nav nav-pills">
    <li role="presentation"><router-link to="/">Home</router-link></li>
    <li role="presentation"><router-link to="/user">User</router-link></li>
  </ul>
</template>
```

از تگ router-link به جای a استفاده میکنیم که به جای فرستادن ریکوئست به سرور سمت کلاینت هندل بشه. To نشون دهنده جاییه که قراره ازش گرفته بشه

```
<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <button @click="navigateToHome" class="btn btn-primary">Go to Home</button>
  </div>
</template>

<script>
  export default {
    methods: {
      navigateToHome() {
        this.$router.push('/');
      }
    }
  }
</script>
```

Make it dynamic

```
export const routes = [
  { path: '/', component: Home },
  { path: '/user/:id', component: User }
];
```

```
export default {
  data() {
    return {
      id: this.$route.params.id
    }
  },
```

```
watch: {
  '$route'(to, from) {
    this.id = to.params.id;
  }
}
```

add a watch to see the changes

```
export const routes = [
  { path: '', component: Home },
  { path: '/user', component: User, children: [
    { path: '', component: UserStart },
    { path: ':id', component: UserDetails },
    { path: ':id/edit', component: UserEdit }
  ] }
];
```

add children routes

How to make it more dynamic?

name: { path: ':id/edit', component: UserEdit, name: 'userEdit' } }

```
<router-link
  tag="button"
  :to="{ name: 'userEdit', params: { id: $route.params.id } }"
  class="btn btn-primary">Edit User</router-link>
```

Query

localhost:8080/user/1/edit?locale=en&q=100

```
:to="{ name: 'userEdit', params: { id: $route.params.id }, query: { locale: 'en', q: 100 } }"
```

Redirecting

* یعنی هرچی!

```
{ path: '/redirect-me', redirect: { name: 'home' } },
{ path: '*', redirect: '/' }
```

```
<router-view name="header-top"></router-view>
<transition name="slide">
  <router-view></router-view>
</transition>
<router-view name="header-bottom"></router-view>
```

Animate it

```
const router = new VueRouter({
  routes,
  mode: 'history',
  scrollBehavior(to, from, savedPosition) {
    if (to.hash) {
      return { selector: to.hash };
    }
    return { x: 0, y: 700 };
  }
});
```

و

```
hash: '#data'
```

اگه توی دیتا یه چیزی تعریف کنیم مثل
توی روت scrollBehavior رو تعریف کنیم که اگه مثلاً آخر روتی که وارد کردیم #data بود بره اونجا، در غیر این صورت بره
به جایی که گفتیم: x=0, y=700

```
router.beforeEach((to, from, next) => {
  console.log('global beforeEach');
  next();
});
```

میتونیم قبل از هر routing اون روت رو چک کنیم و...

```
{ path: ':id', component: UserDetails, beforeEnter: (to, from, next) => {
  console.log('inside route setup');
  next();
} },
```

اگه بخوایم فقط

واسه یه روت خاص چک کنه

```
beforeRouteEnter(to, from, next) {
  if (true) {
    next();
  } else {
    next(false);
  }
}
```

For authentication

به جای true میره مثلا از دیتابیس چک میکنه اطلاعاتی که یوزر داده درسته یا نه و دسترسی داره یا نه

```
beforeRouteLeave(to, from, next) {
  if (this.confirmed) {
    next();
  } else {
    if (confirm('Are you sure?')) {
      next();
    } else {
      next(false);
    }
  }
}
```

حالت دیگه که قبل از ترک کردن صفحه است

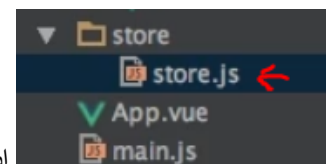
Lazy routing

```
const User = resolve => {
  require.ensure(['./components/user/User.vue'], () => {
    resolve(require('./components/user/User.vue'));
  });
};
```

webpack خودش اینو میفهمه

و درستش میکنه.

Vuex



The screenshot shows a file explorer with a tree view. Under a folder named 'store', there is a file named 'store.js' which is highlighted with a red arrow. Below the 'store' folder, there are two files: 'App.vue' and 'main.js'.

این فایل رو میسازیم و با دستور `npm i -save-dev vuex` میایم vuex رو نصب میکنیم.


```
p.vue × store.js × main.js × index.html × Counter.vue ×
import Vue from 'vue'
import App from './App.vue'

import { store } from './store/store';

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

```
vue × store.js × main.js × index.html × Counter.vue ×
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export const store = new Vuex.Store({
  state: {
    counter: 0
  }
});
```

کلمه `state` کلیدی است و باید از آن استفاده شود. توی کامپوننتمون حالا به راحتی میتونیم بگیریم (تو جایی که متغیر رو عوض

```
export default {
  methods: {
    increment() {
      this.$store.state.counter++;
    },
    decrement() {
      this.$store.state.counter--;
    }
  }
}
```

میکنیم): که استفاده از `$` ضروریه.

و تو جایی که متغیر عوض شده رو میخوانیم ببینیم:

```
export default {
  computed: {
    counter() {
      return this.$store.state.counter;
    }
  }
}
```

حالا اگه بخوایم فانکشن ها رو هم تو همین `store` انجام بدیم میایم و اونا رو تو `getter` تعریف میکنیم:

```
export const store = new Vuex.Store({
  state: {
    counter: 0
  },
  getters: {
    doubleCounter: state => {
      return state.counter * 2;
    }
  }
});
```

```
computed: {
  counter() {
    return this.$store.getters.doubleCounter;
  }
}
```

و اینجوری هم ازش استفاده کنیم:

U can map getters too!


```

<template>
  <div>
    <p>Counter is: {{ doubleCounter }}</p>
    <p>Number of Clicks: {{ stringCounter }}</p>
  </div>
</template>

<script>
  import { mapGetters } from 'vuex';
  export default {
    computed: mapGetters([
      'doubleCounter',
      'stringCounter'
    ])
  }
</script>

```

Mutation!

```

export default {
  methods: {
    increment() {
      this.$store.commit('increment');
    },
    decrement() {
      this.$store.state.counter--;
    }
  }
}

```

```

import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export const store = new Vuex.Store({
  state: {
    counter: 0
  },
  getters: {
    doubleCounter: state => {
      return state.counter * 2;
    },
    stringCounter: state => {
      return state.counter + ' Clicks';
    }
  },
  mutations: {
    increment: state => {
      state.counter++;
    }
  }
});

```

Mutations are synchronous, u cant use them asynchronous

So action!

```

mutations: {
  increment: state => {
    state.counter++;
  },
  decrement: state => {
    state.counter--;
  }
},
actions: {
  increment: ({ commit }) => {
    commit('increment');
  },
  decrement: ({ commit }) => {
    commit('decrement');
  },
  asyncIncrement: ({ commit }) => {
    setTimeout(() => {
      commit('increment');
    }, 1000);
  },
  asyncIncrement: ({ commit }) => {
    setTimeout(() => {
      commit('increment');
    }, 1000);
  }
}
}

```

میتونن به آرگومان اضافه تر هم بگیرن: payload

```

<script>
import {mapActions} from 'vuex';

export default {
  methods: {
    ...mapActions([
      'increment',
      'decrement'
    ])
  }
}
</script>

```

```

mutations: {
  increment: (state, payload) => {
    state.counter += payload;
  },
  decrement: state => {
    state.counter--;
  }
},
actions: {
  increment: ({ commit }, payload) => {
    commit('increment', payload);
  },
  decrement: ({ commit }) => {
    commit('decrement');
  },
  asyncIncrement: ({ commit }) => {
    setTimeout(() => {
      commit('increment');
    }, 1000);
  }
}
}

```

علامت ... syntax مربوط به ES6 هست که میاد اینا رو مپ میکنه و میتونیم راحتتر استفاده کنیم ازشون و همچنین متدهای خودمون

```

@click="increment(100)":
@click="decrement(50)">

```

رو هم اگه لازمه اضافه کنیم. حالا کجاها و چجوری از payload استفاده میکنیم؟

حالا اگه بیشتر از یه آرگومننت داشتیم؟

```
@click="asyncIncrement({by: 50, duration: 500})"
asyncIncrement: ({commit}, payload) => {
  setTimeout(() => {
    commit('increment', payload.by);
  }, payload.duration);
},
```

Input validation

npm i vuedate --save-dev

```
import Vue from 'vue'
import App from './App.vue'
import axios from 'axios'
import Vuelidate from 'vuelidate'

import router from './router'
import store from './store'

Vue.use(Vuelidate)
```

چیزایی که تو validation مینویسیم اسمشون باید با اونی که تو data است و میخوایم validate کنیم یکی باشه. Vuelidate به صورت پیش فرض یه سری چیزا رو داره، میتونیم اگه خواستیم هم خودمون بسازیم.

```
import { required, email, numeric, minValue } from 'vuelidate/lib/validators'
export default {
  data () {
    return {
      email: '',
      age: null,
      password: '',
      confirmPassword: '',
      country: 'usa',
      hobbyInputs: [],
      terms: false
    }
  },
  validations: {
    email: {
      required,
      email
    },
    age: {
      required,
      numeric,
      minVal: minValue(18)
    }
  }
}
```

```
<input
  type="email"
  id="email"
  @input="$v.email.$touch()"
  v-model="email">
```

\$v نمایش دهنده Vuelidate هست. با \$touch() میایم اینا رو به هم

وصل میکنیم. برای ایمیل

```
<input
  type="email"
  id="email"
  @blur="$v.email.$touch()"
  v-model="email">
```

حالت دیگه میتونه این باشه:

```
<div class="input" :class="{invalid: $v.age.$error}">
  <label for="age">Your Age</label>
  <input
    type="number"
    id="age"
    @blur="$v.age.$touch()"
    v-model.number="age">
    <p v-if="!$v.age.minVal">You have to be at least {{ $v.age.$params.minVal.min }}
      years old.</p>
</div>
```

برای عدد

برای `import { required, email, numeric, minValue, minLength, sameAs } from 'vuelidate/lib/validators'`

password, confirm password

```
<div class="input" :class="{invalid: $v.password.$error}">
  <label for="password">Password</label>
  <input
    type="password"
    id="password"
    @blur="$v.password.$touch()"
    v-model="password">
</div>
<div class="input" :class="{invalid: $v.confirmPassword.$error}">
  <label for="confirm-password">Confirm Password</label>
  <input
    type="password"
    id="confirm-password"
    @blur="$v.confirmPassword.$touch()"
    v-model="confirmPassword">
</div>
```

برای

:validator

```
validations: {
  email: {
    required,
    email
  },
  age: {
    required,
    numeric,
    minVal: minValue(18)
  },
  password: {
    required,
    minLen: minLength(6)
  },
  confirmPassword: {
    sameAs: sameAs('password')
  }
},
```

Checkbox

```
<div class="input inline" :class="{invalid: $v.terms.$invalid}">
  terms: {
    required: requiredUnless(vm => {
      return vm.country === 'germany'
    })
  }

```

validator

for arrays

```
<div class="hobbies">
  <h3>Add some Hobbies</h3>
  <button @click="onAddHobby" type="button">Add Hobby</button>
  <div class="hobby-list">
    <div
      class="input"
      v-for="(hobbyInput, index) in hobbyInputs"
      :key="hobbyInput.id">
      <label :for="hobbyInput.id">Hobby #{{ index }}</label>
      <input
        type="text"
        :id="hobbyInput.id"
        @blur="$v.hobbyInputs.$each[index].value.$touch()"
        v-model="hobbyInput.value">
      <button @click="onDeleteHobby(hobbyInput.id)" type="button">X</button>
    </div>
  </div>
</div>

```

```
hobbyInputs: {
  minLength: minLength(1),
  $each: {
    value: {
      required,
      minLength: minLength(5)
    }
  }
}

```

Submit button

```
<div class="submit">
  <button type="submit" :disabled="$v.$invalid">Submit</button>
</div>

```

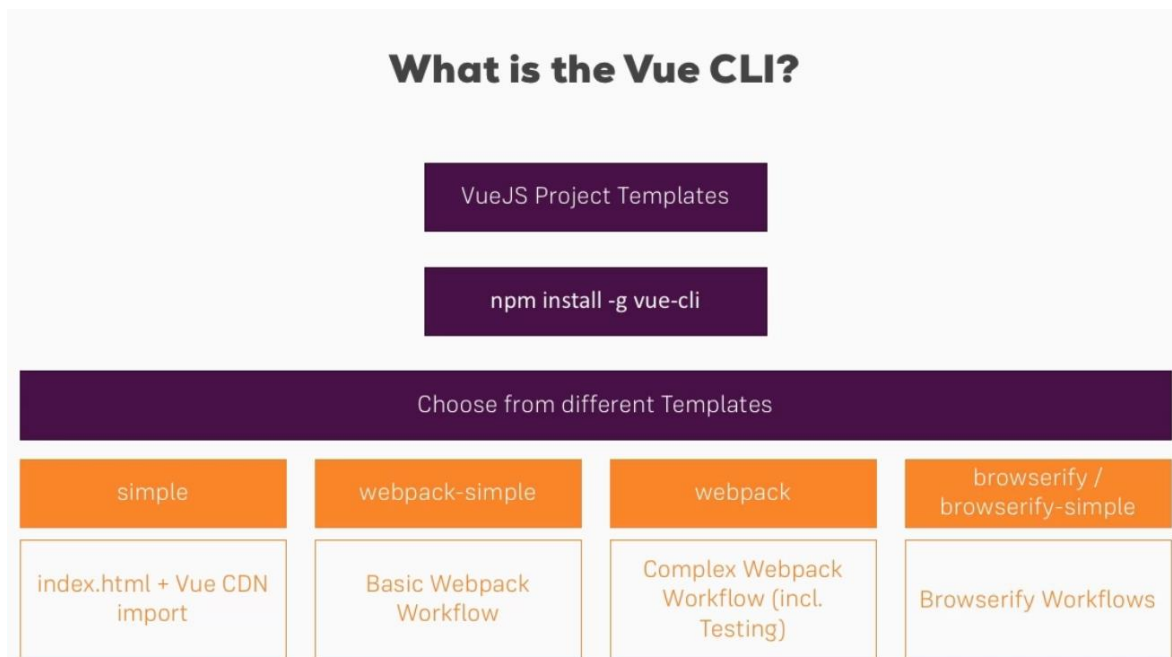
Asynch validator

```
validations: {
  email: {
    required,
    email,
    unique: val => {
      if (val === '') return true
      return axios.get('/users.json?orderBy="email"&equalTo="' + val + '"')
        .then(res => {
          return Object.keys(res.data).length === 0
        })
    }
  }
},

```


یه مدل از validator که خودمون میتونیم بنویسیم. اینجا میاد با توجه به url (به بک اند ربط داره) چک میکنه ببینه ایمیلش تو لیست ایمیل ها هست یا نه. اگه نبود یه آبجکت empty (ربط به بک اند داره همچنان) برمیگردونه یعنی اون شرط جلوی return درست میشه و true برمیگردونه پس میتونه ایمیل رو وارد کنه.

Vue CLI



npm i -g vue-cli

initialize the project and vue-cli is the project name here

```
vue init webpack-simple vue-cli
```

بعدش وارد فولدر پروژه میشیم و dependency هاشو نصب میکنیم و بعدش npm run dev.

توی dependency ها "vue-loader": "^9.7.0"، هست که بهمون اجازه میده SPA ها رو داشته باشیم.

To deploy your app use: npm run build

برای ورژن جدیدترش باید اینجوری نصب کنیم: npm i -g @vue/cli

```
vue create vue-cli-new
```

برای درست کردن اپلیکیشن به جای init دستورش این میشه:

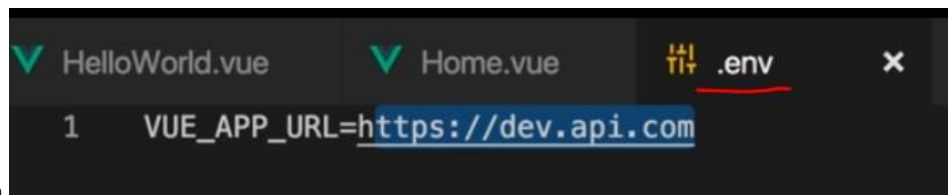
```
Vue CLI v3.0.0-rc.3
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, PWA, Router, Vuex, Linter
? Pick a linter / formatter config: Standard
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? Yes
? Save preset as: 
```

```
npm run serve
```

سپس وارد فولدر پروژه میشیم و

```
vue add vuetify
```

برای اضافه کردن پلاگین ها



```
V HelloWorld.vue V Home.vue .env x
1 VUE_APP_URL=https://dev.api.com
```

وقتی میخوای environment

variable تعریف کنی



```
data() {
  return {
    url: process.env.VUE_APP_URL
  }
}
```

how to use it

برای این که متغیرهایی که تعریف میکنی توی vue code هم قابل استفاده باشن باید اسمشون با VUE_APP_ شروع بشه.