**VueJs** :

Vue یک فریم ورک برای java script است. فریم ورک محیطی است که برای یک زبان برنامه نویسی خاص ایجاد شده که باعث میشه استفاده از اون زبان راحتتر بشه.

vue را به دو صورت میتوان استفاده کرد:

1. برای طراحی قسمتی از وب سایت ( Vue Widgets )
2. برای طراحی کل یک وبسایت به صورت تک صفحه ای ( Vue Website )

روش اول : طراحی در این روش به این صورت هست که ما قالب کلی وبسایت رو با مثلا html به وجود اوردیم و میخایم بخشی از اون رو با استفاده از کامپوننت های ویو طراحی کنیم. مثلا فرم های ثبت نام، میایم با استفاده از ویو کامپوننت این فرم رو طراحی میکنیم و در سایت در جای مخصوص خودش اون رو قرار میدیم.

روش دوم : در این روش وقتی که مثلا سایت رو باز میکنیم برای اولین بار یک درخواست به سرور فرستاده میشه و اطلاعات کلی برای vue ارسال میشه و بعد از اون هر درخواست که در سایت بزنیم ویو اون رو هندل میکنه و دیگه درخواست های بعدی ما سمت سرور نمیره.

**جلسه اول : نصب و راه اندازی** VsCode **(**00 **:** 14**)**

نرم افزار رو از سایت دانلود میکنیم و بعد فایل نصبی رو اجرا میکنیم.

نکته : اگر میخایم بعد از نصب با کلیک راست کردن روی فولدرها بتونیم گزینه vscode هم برای بازکردن فولدر ببینیم باید گزینه های

Add `Open with Code` ….

Add `Open with Code` ….

رو فعال کنیم. و بعد برنامه رو نصب میکنیم.

اکستنشن های لازم برای vue :

1. Live Server : برای اینکه خروجی رو به صورت لایو ببینیم. برای این کار دکمه های alt + l + o رو میزنیم تا برنامه اجرا بشه و بعد از سیو کردن هر تغییر تغییرات به صورت اتوماتیک اعمال خواهد شد.
2. Vetur

مرحله بعد نصب ویو روی پروژه هست.

دو روش وجود داره برای نصب وجود داره cdn و npm. در اینجا ما ازcdn استفاده میکنیم و فایل cdn رو به پروژه اضافه میکنیم.

 <script src="https://unpkg.com/vue@next"></script>

فایلی که در سایت برای استفاده از ویو هست به صورت بالاست ولی ادرس بالا اشاره میکنه به اخرین ورژن ویو ولی چون میخایم از یک ورژن خاص استفاده کنیم کلمه next رو برمیداریم و ورژن 3.0.2 رو به جاش قرار میدیم.

**جلسه دوم : ساخت اپ توسط** Vue **(**00 **:** 22**)**

وقتی که کتابخانه ویو رو به پروژه اضافه کردیم یک ابجکت در اختیار ما قرار میده به نام Vue که میتونیم برای ساخت کامپوننت ازش استفاده کنیم.

const app = Vue.createApp()

بعد از اینکه کامپوننت رو ساختیم باید بگیم کجا باید قرار بگیره. مثلا میخایم داخل المنتی با ایدی app این کامپوننت قرار بگیره.

app.mount("#app")

داخل createApp یک سری پروپرتی پیش فرض وجود داره که میتونیم ازشون استفاده کنیم. Data, function , template

Template برای استفاده از کدهای html در ویو استفاده میشه مثلا:

const app = Vue.createApp({

    template: '<h2>this is a test text</h2>'

})

نکته : زمانی که برنامه رو اجرا میکنیم المنتی که ویو بهش مانت شده یک مقدار بهش اضافه میشه با نام:

data-v-app

که نشون میده هرچیزی که داخل این المنت نوشته میشه توسط ویو مدیریت میشه.

**جلسه سوم : بررسی** DataTemplate **(**00 **:** 29**)**

میخوایم یک سری متغیرها ایجاد کنیم مقادیری داخلش ذخیره کنیم و اون متغیرها رو توسط ویو مدیریت کنیم. برای این کار از پروپرتی data استفاده میکنیم. که یک ابجکت بهمون برمیگردونه

const app = Vue.createApp({

    data(){

        return{

             title: "bofkoor",

             author: " sadegh hedayat",

             age: 45

        }

    }

})

برای نمایش این متغیرها به صورت زیر عمل میکنیم :

 <div id="app">

        <p>{{title}} -- {{author}} -- {{age}}</p>

 </div>

نکته : اگر این متغیرها رو خارج از المنتی که توسط ویو مدیریت میشه بنویسیم دیگه مقدار متغیر برگشت داده نمیشه. **پس این روش توسط ویو شناسایی میشه.**

**جلسه چهارم : بررسی متدها و ایونت کلیک (**00 **:** 35**)**

مثال : میخایم دوتا کلید درست کنیم که با کلیک کردن روشون سنی که در مثال بالا زدیم کم و زیاد بشه. برای این کار از دایرکتیو v-on استفاده میکنیم.

<button v-on:click="age++">Increase</button>

<button v-on:click="age--">Increase</button>

نکته : به جای استفاده از v-on: میتونیم از @ استفاده کنیم.

همچنین میتونیم بعد از کلیک، متدی رو صدا بزنیم:

<div @click="changeTitle('sag velgard')">changeTitle</div>

اگر متد ورودی نداشته باشه میتونیم پرانتز جلوی متد هم نزاریم. بعد:

methods:{

        changeTitle(abc){

            this.title = abc        }    }

نام متغیر ورودی دلخواه هست. اگر کلمه this رو نزاریم نمیتونیم به مقدار title دسترسی داشته باشیم.

**جلسه پنچم : رندر شدن های شرطی** ConditionalRendering **(**00 **:** 46**)**

گاهی اوقات لازم هست که بودن یا نبودن یک المنت رو شرطی کنیم. مثلا میخوایم اگر یک کاندیشن true شد کتاب مثال بالا رو نمایش بدیم و اگر false شده کتاب رو نمایش ندیم.

برای این کار از دایرکتیو v-if استفاده میکنیم.

 <button @click="ChangeShowBook">

        <span v-if=showBooks>hide books</span>

        <span v-if=!showBooks>show books</span>

 </button>

و کدهای ویو:

const app = Vue.createApp({

    data(){

        return{

             showBooks: true

        }

    },

    methods:{

        ChangeShowBook(){

            this.showBooks = !this.showBooks;

        }

    }

})

همچنین میتونیم به جای v-if دومی که استفاده کردیم از v-else استفاده کینم:

 <button @click="ChangeShowBook">

        <span v-if=showBooks>hide books</span>

        <span v-else>show books</span>

 </button>

نکته: گاهی وقت ها به جای استفاده از v-if میتونیم از v-show استفاده کنیم. تفاوت این دو روش به این صورت هست که اگر v-if، false باشه خود المان رو از صفحه حذف میکنه و اگر true باشه المان رو برای ما به وجود میاره و این حذف و اضافه کردن المان برای ما هزینه داره. ولی v-show میاد از یک خاصیت css برای نمایش و عدم نمایش استفاده میکنه. اگر کاندیشنش false بود میاد display المان رو none میکنه و اگر کاندیشن true باشه display رو برای المان برمیداره.

**جلسه ششم : بررسی ایونت های موس (00 : 57)**

چهار ایونت معروف موس :

mouseover : وقتی که با موس روی اون المنت میریم اتفاق مورد نظر رخ میده.

mouseleave : وقتی که موس از محدوده المنت خارج میشه اتفاق مورد نظر رخ میده.

dblclick : وقتی که بر روی المنت دبل کلیک کنیم اتفاق مورد نظر رخ میده.

mousemove : موس به ازای هر جابه جایی که داخل المنت داشته باشه اتفاق مورد نظر رخ میده.

نکته : وقتی که ایونتی اتفاق میافته ابجکتی شامل محتوای اون ایونت به عنوان ابجکت به متد ما ارسال میشه که میتونیم اون مقادیر رو بگیریم. برای گرفتن این مقادیر هم کافیه در ورودی متد یک متغیر تعریف کنیم و ازش مقادیر رو بگیریم.

مثال: میخوایم با استفاده از ایونت mousemove محتوای offsetx رو تغییر بدیم:

<div class="box" @mousemove="handleEvent">{{x}} -- {{y}}</div>

handleEvent(e){

            this.x = e.offsetX;

            this.y = e.offsetY

        }

**جلسه هفتم : نمایش مقادیر یک لیست** v-for **(**01 **:** 08**)**

برای حلقه روی لیست ها به کار برده میشود. پیاده سازی به صورت زیر:

 <ul v-for="book in books">

          <li>

            <h3>{{book.title}}</h3>

            <p>{{book.outhor}}</p>

          </li>

</ul>

و :

data(){

        return{

             books: [

                {title: "bofkor", outhor: "sadegh hedayat"},

                {title: "sag velgard", outhor: "sadegh hedayat"},

                {title: "kelidar", outhor: "mohamad davodi"}

             ]

        }

    },

**جلسه هشتم : اتصال ویژگی ها** AttributeBinding **(**01 **:** 15**)**

در این جلسه میخایم از متغیرهایی که داخل فایل vuejs درست کردیم داخل اتریبیوت های هر المنت استفاده کنیم. برای این کار از دایرکتیو v-bind استفاده میکنیم. و قبل از هر اتریبیوتی که میخایم این ویژگی رو داشته باشه v-bind: رو قرار میدیم. bind به معنی پیوند دادن دو چیز به هم میباشد.

مثلا میخایم مقدار url یک تگ a رو از متغیری که در vue تعریف کردیم بگیریم :

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

نکته : میتونیم به جای استفاده از v-bind: از : خالی هم به صورت خلاصه شده استفاده کنیم و دوباره همان عملکرد رو دریافت کنیم.

**جلسه نهم : کلاس های پویا** Dinamicclasses **(**01 **:** 21**)**

یعنی اینکه گاهی اوقات برای هر المنت کلاس ایجاد میکنیم و بعد برای اون کلاس استایل هایی رو در نظر میگیریم اینجا میخایم ایجاد شدن هر کلاس برای هر المنت با توجه به یک کاندیشن باشه. که در vue هندل میشه.

مثلا لیستی داریم که با v-for میخواهیم نمایش دهیم اما بعضی از مقادیر این لیست باید کلاس خاصی رو بگیرند.

<ul v-for="book in books">

          <li :class="{fav: book.isfav}">

            <h3>{{book.title}}</h3>

            <p>{{book.outhor}}</p>

          </li>

</ul>

برای این کار اول برای ابجکت هایی که در vue تعریف کردیم یک متغیر دیگر در نظر میگیریم مثلا به نام isfav که مقادیر true و false میتونن داشته باشن و هرکدوم از اونها که مقدار true داشته باشن کلاس fav به اونها اضافه میشه.

**جلسه دهم : چالش – اضافه کردن به** fav **(**01 **:** 30 **)**

میخایم با کلیک کردن روی هریک از المنت های یک لیست استایل خاصی به اون ابجکت لیست اضافه بشه.

<ul v-for="book in books">

          <li @click="changeStyle(book)" :class="{fav: book.isfav}">

            <h3>{{book.title}}</h3>

            <p>{{book.outhor}}</p>

          </li>

</ul>

حالا در vue یک متد به نام changeStyle مینویسیم و book رو در ورودی میگیریم و بعد مقدار isfav رو به برعکس حالتی که الان داره تغییر میدیم.

**جلسه یازدهم : بررسی** ComputedProperties **(**01 **:** 34**)**

داخل این پروپرتی مانند methode نیز متدهایی رو میتونیم بنویسیم ولی با این تفاوت که این متدها زمانی اجرا میشوند که دیتاهای داخل کامپوننت تغییر کنند. مثلا فرض کنید مانند مثال بالا با کلیک روی هریک از ابجکت های لیست مقدار isfav که تغییر کرد به false اون ابجکت هم دیگه نمایش داده نشه.

<ul v-for="book in filterdBooks">

          <li @click="changeStyle(book)" :class="{fav: book.isfav}">

            <h3>{{book.title}}</h3>

            <p>{{book.outhor}}</p>

          </li>

</ul>

و:

 computed: {

    filterdBooks() {

      return this.books.filter(book => book.isfav)

    },

**جلسه دوازدهم : علت استفاده از** VueCLI **(**01 **:** 40**)**

میخایم مدیریت کل وبسایتمون در اختیار vue باشه. ما اگر بخایم از vue برای مدیریت کل وبسایتمون استفاده کنیم باید از Vue CLI یا همون (vue command line interface) استفاده کنیم.

**جلسه سیزدهم : طریقه استفاده از** VueCLI **(**01 **:** 46**)**

برای اینکه بتونیم پکیج vue cli رو نصب کنیم باید پکیج Node رو داخل سیستم نصب داشته باشیم. ما از node استفاده میکنیم تا بتونیم از پکیچ منیجر یا به اختصار npm استفاده کنیم برای نصب پکیج هایی که لازم داریم.

ابتدا به سایت <https://nodejs.org> میریم و لینک دانلود برنامه رو دریافت میکنیم و node رو نصب میکنیم. برای اینکه مطمعن بشیم node نصب شده داخل cmd دستور node –v رو میزنیم و اگر نصب شده باشه ورژن node که نصب شده به ما نمایش داده میشه.

حالا :

npm install –g @vue/cli

نکته : -g به این خاطر گذاشتیم که پکیج vue cli به صورت گلوبال نصب بشه و در هر جایی از سیستم که خاستیم بتونیم از پکیج های vue استفاده کنیم.

حالا میخایم پروژه خودمون رو ایجاد کنیم و ابتدا باید مسیری که میخایم پروژه درش نصب بشه رو مشخص کنیم.

بعد از مشخص کردن مسیر دستور vue create project-1 رو مینویسیم.

Project-1 نام پروژه ما هست. مرحله بعدی انتخاب ساختار پروژه هست.

میتونیم ساختار پیش فرض vue2 یا vue3 رو انتخاب کنیم یا اینکه ساختار رو به صورت دستی تنظیم کنیم. مرحله بعد یک سری گزینه با توجه به نیاز میتونیم فعال یا غیر فعال کنیم.

در مرحله بعد نسخه ویو رو انتخاب میکنیم. و در اخر میپرسه که میخایم این تنظیمات رو برای پروژهای بعدی هم داشته باشی یا نه.

**جلسه چهاردهم : بررسی فایل ها و فولدرهای پروژه (**01 **:** 54**)**

فولدر node\_modules : یک سری ماژول ها و پکیج های درونش هست که پروژه ازش استفاده میکنه.

فولدر public : که دو فایل داخلش هست، favicon.ico و index.html فایل اولی یک ایکونی هست که وقتی پروژه ران بشه بالای سایت به نمایش درمیاد و فایل دوم ساختار کلی html ما هست و داخلش یک المنت div با ایدی app هست که قرار هست ساختاری که با vue میسازیم روی این المنت قرار بدیم.

فولدر src : که داخلش دوتا فولدر و دوتا فایل هست. فولدر assets داخلش یک تصویر هست که در مرکز پروژه مون قرار داره و میتونیم داخلش چیزهای جانبی مثل فونت ها تصاویر و ... رو قرار بدیم.

فولدر components که کامپوننت هایی که ایجاد میکنیم و مدیریتشون توسط ویو انجام میشه داخل این فولدر قرار میگیره.

فایل main.js یک فایل جاوا اسکریپت هست که ویو رو به المنت app اضافه میکنه.

فایل App.vue یک فایل کامپوننت هست که کامپوننت اصلی ما هست و کامپوننت های که زیرمجموعه وب سایت ما هستند میره داخل فولدر components .

کامپوننت ما سه قسمت داره، template ، script و style.

فایل babel.config.js برای اینکه ساختارهای جدیدی که vue داره و مرورگرها اون رو پشتیبانی نمیکنند اون ها رو به نسخه های قدیمی تبدیل کنه.

فایل های package که نیاز نیست چیزی رو به صورت دستی اونجا ویرایش کنیم و به صورت اتوماتیک تمام پکیج هایی که پروژه ما داره رو برامون قرار میده.

و مرحله بعد برای اجرا کردن پروژه از این دستور استفاده میکنیم:

npm run serve

**جلسه پانزدهم : بررسی فایل ها و تگ** Template **(**02 **:** 06**)**

Script و style تگهای هستند که در vue اختیاری هستند ولی template یک تگ اجباری است و هر فایل vue.js باید حتما این تگ رو داشته باشه.

نکته : اگر پروژه ای رو از جایی دریافت کردیم قبل از اینکه پروژه را اجرا کنیم از دستور npm install استفاده میکنیم تا خود dependencies هایی که در package.json وجود داره دانلود و نصب کنه تا بتونیم از پروژه استفاده کنیم.

زمانی که داخل template در فایل App.vue میریم :

 <img alt="Vue logo" src="./assets/logo.png">

<HelloWorld msg="Welcome to Your Vue.js App"/>

که یک تصویر هست و یک کامپوننت که باقی صفحه در یک کامپوننت پیاده سازی شده است.

زمانی که میخایم از یک کامپوننت استفاده کنیم باید اون کامپوننت رو improt کنیم :

import HelloWorld from './components/HelloWorld.vue'

export default {

  name: 'App',

  components: {

    HelloWorld

  }

}

**!! تمرین :** میخایم متن this is a test text رو با استفاده از vue cli در صفحه چاپ کنیم.

<template>

  <h1>{{title}}</h1>

</template>

و:

data(){

    return{

      title: "this is a test text"

    }

  }

قسمت بعدی style هست که میتونیم به تگهایی که در template هست استایل بدیم.

نکته : این استایل ها داخل فایل یا کامپوننت اصلیمون به وجود اومده و هر کامپوننتی که بعدا ایجاد کنیم چون این فایل root هست این استایل به صورت پیش فرض روی همه اونها اعمال میشه که در ادامه این ویژگی رو غیر فعال میکنیم.

**جلسه شانزدهم : دریافت و کنترل تگ با** TemplateRefs **(**02 **:** 15**)**

زمانی که در جاوا اسکریپت میخاستیم مقدار یک المنت رو بگیریم از ایدی یا کلاس یا نام المنت میتونستیم استفاده کنیم ولی زمانی که داخل کامپوننت هستیم نمیتونیم از اون حالت ها استفاده کنیم و باید از یک حالت دیگه استفاده کنیم به نام Template Refs.

مثال : فرض کنیم یک اینپوت فرم داریم و یک دکمه که زمانی که روی دکمه زده شده اینپوت رو داخل کنسول به نمایش بزاریم:

<input type="text" ref="name">

  <button @click="handleEvent">click</button>

و:

  methods: {

    handleEvent(){

      console.log(this.$refs.name);

    }

  }

حالا زمانی که روی دکمه زده بشه در کنسول تگ اینپوت رو میتونیم ببینیم.

توجه : زمانی که روی دکمه بزنیم خود تگ اینپوت رو داریم نه مقداری که در اینپوت وجود داره.

!! تمرین : میخایم در مثال بالا زمانی که اینپوت رو به دست اوردیم میخایم کلاس hasan رو به اینپوت اضافه کنیم.

handleEvent(){

      this.$refs.name.classList.add("hasan")

    }

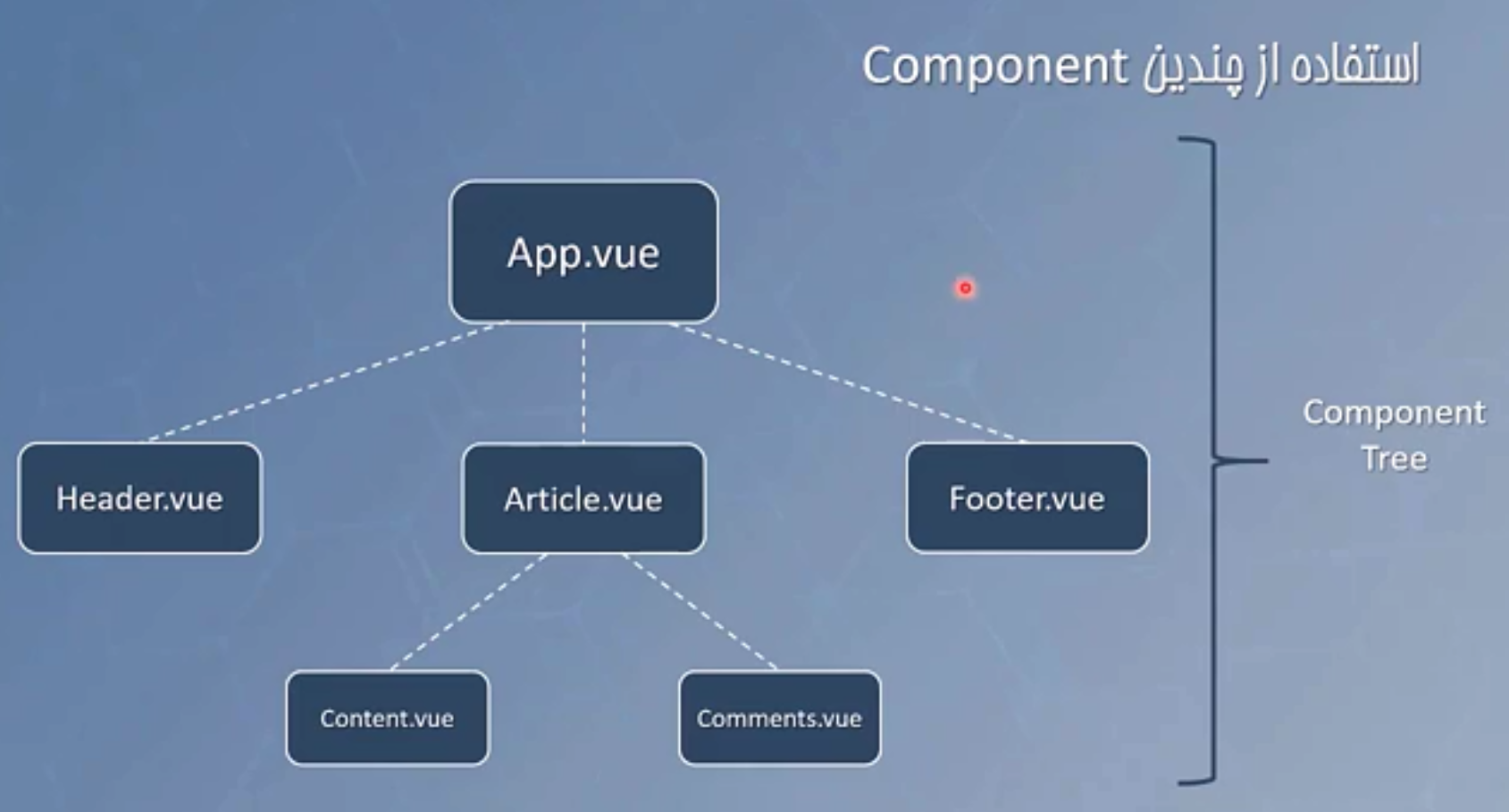
**جلسه هفدهم : استفاده از چندین** Component **(**02 **:** 22**)**

چرا از کامپوننت های مختلف استفاده میکنیم: به این دلیل که اگر همه کد ها رو داخل app بنویسیم درهمریختگی و نامنظمی زیادی به وجود می اید.

و دومین دلیل اینکه میتونیم از این کامپوننت ها چندین بار استفاده کنیم.

App.vue کامپوننت اصلی ما هست و برای اینکه بخایم اپلیکیشن رو به قسمت های مختلف تقسیم کنیم باید کامپوننت های دیگه رو زیرمجموعه این کامپوننت در نظر بگیریم.

به صورت زیر:



همچنین میتونیم برای کامپوننت های زیر مجموعه نیز کامپوننت های زیرمجموعه ای رو اضافه کنیم. و به کل این کامپوننت ها component Tree یا ساختار به صورتی درختی گفته میشه.

نکته : برای نام گزاری فایل های کامپوننت یا فایل های ویو بهتره نام فایل با حرف بزرگ شروع بشه.

مثال : میخایم زمانی که سایت اجرا میشه یک مدال باز بشه و متن modal content رو به ما نمایش بده.

اول باید کامپوننت Modal رو داخل فولدر component ایجاد کنیم و محتوای زیر رو داخلش مینویسیم:

<template>

  <div class="backdrop">

    <div class="modal">

        <p>modal content</p>

    </div>

  </div>

</template>

کلاس هایی که به دیو ها داده شده برای استایل دهی به اونها استفاده شده تا مدال ما ظاهر بهتری داشته باشه.

در مرحله بعد باید این کامپوننت رو داخل app.vue استفاده کنیم. اول کامپوننت رو improt میکنیم.

import Modal from "./components/Modal.vue";

بعد اون رو ریجستر میکنیم:

components: { Modal },

و در اخر هم در template ازش استفاده میکنیم:

  <Modal />

**جلسه هجدهم : استایل و استایل گلوبال** Component **(**02 **:** 35**)**

وقتی که ما برای یک المنت در یک کامپوننت والد استایلی را در نظر میگیریم مشکلی که وجود دارد اگر در کامپوننت چایلد از اون المنت استفاده کنیم اون استایل ها روی این المنت ها هم اعمال میشه.

و اگر در کامپوننت چایلد برای اون المنت یک استایل دیگه در نظر بگیریم. ولی اتفاقی که می افته این هست که هر دو استایل با هم ترکیب میشن و برای کامپوننت والد و فرزند در نظر گرفته میشن.

حالا اگر بخایم استایلی که در کامپوننت فرزند هست جای دیگری استفاده نشه یکی از راه ها این هست که به صورت زیر style رو تغییر بدیم:

<style scoped>

</style>

وقتی میایم از scoped استفاده میکنیم تمام استایل های که مینویسیم فقط داخل همون کامپوننت استفاده میشن.

راه دیگه ای که هست اینه که برای المان ها کلاس های مختلف در نظر بگیریم و با صدا زدن نام کلاس به المان ها استایل بدهیم.

نکته : اگر بخایم یک سری استایل رو به صورت کلی به تمام پروژه بدیم میام داخل فولدر assets یک فایل درست میکنیم ( مثلا با نام global.css ) و اینجا این استایل های سراسری رو بنویسیم.

بعد این فایل رو داخل main.js این فایل رو improt میکنیم.

import './assets/gloabl.css'

**جلسه نوزدهم : ارسال داده از کامپوننت با** Props **(**02 **:** 45**)**

بعضی مواقع نیاز هست که اطلاعاتی رو از کامپوننت والد به کامپوننت فرزند ارسال کنیم. مثلا در مثال بالا میخایم اطلاعاتی که در مدال نمایش میدیم رو از کامپوننت App.vue بگیریم.

اول در کامپوننت app.vue مقادیری که میخایم در مدال بگیریم رو در جایی که مدال را صدا زدیم تعریف میکنیم.

 <Modal header="sign up for a giveaway" text="Grab the course for half price"/>

بعد در کامپوننت مدال با استفاده از پروپرتی Props :

props: ['header', 'text']

و در اخر هم به روش قبل اون رو هر جایی از کامپوننت مدال که خواستیم صدا میزنیم.

نکته: اگر ما بخایم از این روش استفاده کنیم زمانی که مقادیر که به کامپوننت فرزند میخایم ارسال کنیم زیاد باشه به مشکل میخوریم و کار درستی نیست. و روش بهتر این هست که تمام این متغیر ها رو در قالب یک پروپرتی به کامپوننت ارسال کنیم:

 data() {

    return {

      header: "sign up for a giveaway",

      text: "Grab the course for half price"

    };

  },

و:

  <Modal :header="header" :text="text"/>

!! تمرین : میخایم برای مدالی که در بالا درست کردیم چند تم مختلف در نظر بگریم و در جاهای مختلف از تم مخصوص به همان جا استفاده کنیم.

اول باید در کامپوننت های والدی که میخان از کامپوننت مدال استفاده کنن به این صورت عمل کنیم:

  <Modal :header="header" :text="text" :theme="sale"/>

مقدار اخر (theme ) مربوط به تمی هست که میخایم مدال باهاش باز بشه. و در هر کامپوننت با توجه به کارایی میتونه مقدار مختلفی رو داشته باشه. بعد در کامپوننت مدال :

    <div class="modal" :class="{sale: theme === 'sale'}">

میگیم که اگر theme برابر با sale بود برو استایل هایی رو که برای کلاس sale در نظر گرفتیم اجرا کن. در اینجا فقط برای sale نوشتیم ولی میتونیم این شرط رو برای چند کلاس بنویسیم و هر کلاس هم استایل مخصوص خود رو داشته بشه.

**جلسه بیستم : انتشار ایونت های سفارشی با** emit **(**02 **:** 53**)**

میخایم در ادامه مثال بالا مدال زمانی باز بشه که یک کلید در صفحه اصلی فشار داده بشه و اگر در اطراف مدال کلیک کردیم مدال از بین بره.

برای قسمت اول به این صورت عمل میکنیم:

 <div v-if="showModal">

  <Modal :header="header" :text="text" :theme="sale" @close="showmodal"/>

  </div>

  <button @click="showmodal">Show Modal</button>

و در قسمت متدها :

    showmodal(){

      this.showModal = !this.showModal;

    }

ولی برای قسمت دوم زمانی که مدال باز هست به این کامپوننت دسترسی نداریم تا بتونیم از متد showmodal استفاده کنیم و در واقع باید از یک واسط استفاده کنیم که از کامپوننت فرزند یک چیزی شبیه به ایونت برای ما برگردونه و ما با استفاده از اون ایونت بیام و این متد رو صدا بزنیم:

 <div class="backdrop" @click="closeModal">

و :

 methods: {

        closeModal(){

            this.$emit('close')

        }

    }

**جلسه بیست و یکم : اصلاح کنند های ایونت (**03 **:** 02**)**

مشکلی که در مثال قبل وجود داشت این بود که زمانیکه مدال ظاهر میشد اگر روی خود مدال هم کلیک میکردیم مدال محو میشد و برای رفع این مشکل از ایونت مدیفایر ها استفاده میکنیم.

میخایم بگیم که برای اینکه مدال محو بشه کلیک تنها کافی نیست و مثلا یک کلیدی هم باید فشار داده بشه.

<div class="backdrop" @click.self="closeModal">

این باعث میشه که اگر خارج از مدال کلیک کردیم متد closeModal اجرا بشه.

**جلسه بیست و دوم : بررسی** Slots **(**03 **:** 06**)**

گاهی اوقات میخایم یک سری تمپلیت از کامپوننت والد به کامپوننت فرزند ارسال کنیم یعنی مثلا چندین تگ ارسال کنیم در این مواقع نمیتونیم از props استفاده کنیم. و باید از Slots استفاده کنیم:

<Modal @close="showmodal">

    <h1>Open Code Giveaway</h1>

    <p>Grab the course for half price</p>

  </Modal>

و بعد در کامپوننت مدال :

  <div class="modal">

    <slot></slot>

    </div>

حال فرضا میخایم که دوتا تمپلیت برای مدال ارسال کنیم در اینجا باید از تگ تمپلیت به صورت زیر استفاده کنیم:

   <template v-slot:links>

      <a href="#">Sign up</a>

      <a href="#">More Info</a>

    </template>

برای slot یک نام براش در نظر میگیرم و در کامپوننت مدال :

 <div class="action">

      <slot name="links"></slot>

    </div>

با مشخص کردن نام slot گفتیم که باید از کدام تمپلیت استفاده کند.

**جلسه بیست و سوم : چالش – استفاده مجدد از کامپوننت (**03 **:** 17**)**

دقیقا مثل روش اولی که از کامپوننت استفاده کردیم دوباره میتونیم کامپوننت رو صدا بزنیم و مجداا از کامپوننت استفاده کنیم.

**جلسه بیست و چهارم : بررسی** Teleport **(**03 **:** 22**)**

زمانی که برنامه اجرا میشه کدهای ویو ای که زدیم روی المنتی قرار میگیره با ایدی app حالا میخایم زمانی که این اتفاق داره میفته بیایم یک سری از تمپلیت هامون رو روی یک المنتی دیگری که ایدی app نداره قرار بدیم.

اول از همه در فایل index.html که در فولدر public قرار داره المان مورد نظر رو ایجاد میکنیم:

   <div id="app"></div>

    <div class="modals"></div>

حالا تمپلیت هایی که قرار هست در المنت با کلاس modal قرار بگیرند به صورت زیر عمل میکنیم :

  <teleport to=".modals" v-if="showModal">

  <Modal @close="showmodal">

    <h1>Open Code Giveaway</h1>

    <p>Grab the course for half price</p>

    <template v-slot:links>

      <a href="#">Sign up</a>

      <a href="#">More Info</a>

    </template>

  </Modal>

  </teleport>

حالا اگر پروژه رو اجرا کنیم تغییر رو نمیبینم اما وقتی که کدها رو نگاه میکنیم یک المنت داریم با کلاس modals که چیزی داخلش نیست و به محض اینکه دکمه نمایش مدال رو میزنیم و کدهای بالا اجرا میشن تمامی تگهایی که در بین تگ teleport قرار دارند داخل این المنت هم اضافه میشن.

مشکلی که وجود داره این هست که زمانی که مدال نمایش داده میشه استایل هایی که برای المنت app قرار داره روی المنت ما اعمال نشده و برای اینکه اون استایل ها رو داشته باشیم کافیه در جایی که داریم به المنت با ایدی app استایل میدیم کلاس modal هم اضافه کنیم.

**جلسه بیست و پنچم : پیش نمایش و راه اندازی پروژه (**03 **:** 27**)**

میخایم ریکشن تایمر درست کنیم که یک دکمه داریم به نام paly که به محض اینکه روش کلیک میکنیم منتظر میمونیم تا یک باکس برامون باز بشه بعد که باکس ظاهر شد زمان شروع میکنه به سپری شدن و با توجه به زمانی که سپری شده تا ما ریکشن نشون بدیم و روی باکس کلیک کنیم زمان ریکشن رو به ما نشون میده و با توجه به سرعت عملون یک متن هم بهمون نمایش میده.

پروژه خام رو ایجاد میکنیم. دو کامپوننت با نام های Block و Results ایجاد میکنیم که بعد ازش استفاده خواهیم کرد.

**جلسه بیست و ششم : شروع یک بازی جدید (**03 **:** 33**)**

توی این قسمت میام دکمه و اون باکس رو طراحی میکنیم و تاخیری که برای نمایش باکس رو داریم به وجود میاریم همچنین زمانی که روی دکمه زده شد و باکس نمایش داده شد دکمه غیرفعال میشه.

<template>

  <h1>OpenCode Reaction Timer</h1>

  <button @click="start" :disabled="isPlaying">Paly</button>

  <Block v-if="isPlaying" :delay='delay'/>

</template>

<script>

import Block from "./components/Block.vue";

export default {

  name: "App",

  components: { Block },

  data() {

    return {

      delay: null,

      isPlaying: false,

    };

  },

  methods: {

    start() {

      this.delay = 2000 + Math.random() \* 5000;

      this.isPlaying = true;

    },

  },

};

</script>

Math.random عددی رندم بین صفر تا یک رو به ما میده و ما در 5000 ضرب کردیم تا عددی بین 0 تا 5 ثانیه رو داشته باشیم همچنین در حالت دیفالت حداقل مقدار تاخیر 2 ثانیه هست که با مقدار رندم جمع شده. و حالا کامپوننت Block :

<template>

  <div class="block">

    click me

  </div>

</template>

<script>

export default {

    props: ['delay']

}

</script>

**جلسه بیست و هفتم : چرخه حیات** Component **(**03 **:** 45**)**

ما برای ایجاد هر کامپوننت یک lifesicll یا چرخه حیات طی میکنیم که در هر مقطعی یک حالت به خصوصی داره و به هر کدوم از اون حالت ها هوکس گفته میشه.

دلیل به وجود امدن هوکس : به این دلیل هوکس به وجود امده که ما در هر مقطعی از این چرخه کار دلخواهمون رو خیلی راحتر انجام بدیم. مثلا میخایم کامپوننتمون رو همون لحظه ای که داره اجرا میشه یک کار به خصوصی انجام بدیم یا وقتی که کامپوننت کامل ایجاد شده بخایم یه کار به خصوصی انجام بدیم.

هوکس هایی که داریم :

زمانی که کامپوننتمون ایجاد میشه. هوکسی که در این مقطع داریم Before Create هست. توی این سطح دسترسی به دیتا و تمپلیت نداریم.

زمانی که روی یک المنت به خصوص مانت میشه. هوکسی که در این مقطع داریم Created هست. و سطح دسترسی ما به دیتا باز میشه. ولی قابلیت استفاده از ایونت و تمپلیت ها رو نداریم.

زمانی که ساختارهای داخلش میاد و ابدیت میشه. هوکسی که در این مقطع داریم Before Mount هست. که در این سطح دسترسی ما به دیتا، ایونت و تمپلیت ها باز میشه.

زمانی که کامپوننتمون داره نابود میشه. و در این مقطع هوکس mounted رو میبینیم. این مقطع برای رکوئست زدن به سرور بهترین زمان هست.

مثال:

mounted() {

    console.log("component");

    setTimeout(() => {

      this.showBlock = true;

    }, this.delay)

  },

  updated(){

    console.log(this.delay);

  },

  unmounted(){

    console.log("component unmounted");

  }

زمانی که کامپوننت صدا زده میشه هوکس mounted صدا زده میشه و متغیر showBlock بعد از تاخیری که در جلسه پیش تعریف کردیم مقدارش ture میشه و به محض اینکه ویرایشی داخل کامپوننت اتفاق بیفته هوکس update اتفاق میفته. و زمانی که استفاده از این کامپوننت تمام بشه هوکس unmounted هم اتفاق میفته.

**جلسه بیست وهشتم : ایجاد تایمر واکنش (**03 **:** 57**)**

حال میخایم مدت زمانی که باکس ظاهر میشه تا زمانی که روش کلیک کنیم این مدت زمان رو به دست بیاریم

<div class="block" v-if="showBlock" @click="stopTimer">click me</div>

و:

  methods:{

    startTimer(){

      this.timer = setInterval(() => {

        this.reactionTimer +=10;

      },10)

    },

    stopTimer(){

      clearInterval(this.timer)

      console.log(this.reactionTimer)

    }

  },

  mounted() {

    console.log("component");

    setTimeout(() => {

      this.showBlock = true;

      this.startTimer();

    }, this.delay)  },

زمانی که کامپوننت ما ظاهر میشه startTimer رو در هوکس mounted صدا میزنیم و در این تابع با متد setInterval مدت زمانی که میگذره رو داخل متغیر reactionTimer میریزیم بعد که روی باکس کلیک شد متد stopTimer صدا زده میشه و در اون متد با استفاده از دستور clearInterval تایمر رو متوقف میکنیم و تایمی که گذشته تا روی باکس کلیک بشه در کنسول نمایش میدیم.

**جلسه بیست و نهم : ارسال ایونت های سفارشی با داده (**04 **:** 06**)**

نکته : برای اینکه بتونیم از دیتای یک کامپوننت فرزند در یک کامپوننت فرزند دیگه استفاده کنیم باید ابتدا دیتا رو به کامپوننت والد ارسال کنیم و بعد از اونجا بفرستیم به کامپوننت مورد نظر.

  this.$emit('end', this.reactionTimer)

به همراه emit میتونیم مقداری رو به کامپوننت والد ارسال کنیم که این مقدار رو میتونیم در ورودی متدی که در کامپوننت والد نوشتیم دریافت کنیم.

**جلسه سی ام : چالش – نمایش موارد با** ResultsComponent **(**04 **:** 16**)**

در این جلسه مقداری که در reactionTimer داریم رو به کامپوننت results میفرستیم و اونجا نمایشش میدیم.

**جلسه سی و یکم : ایجاد کردن یک واکنش متن به پروژه (**04 **:** 20**)**

توی این جلسه میخایم با توجه به زمانی که گذشته و ریکشنی که داشتیم یک متی رو به کاربر نشون بدیم.

mounted(){

        if(this.score < 250){

            this.rank = "عالی بود"

        }else if(this.score <400){

            this.rank = "هی بد نبود"

        }else{

            this.rank = "حلزون کی بودی"

        }

    }

**جلسه سی و دوم : معرفی و راه اندازی فرم ها (**04 **:** 25**)**

ابتدا یک کامپوننت درست میکنیم برای فرم به نام Signupform و یک فرم داخلش ایجاد میکنیم.

و داخل کامپوننت روت اون رو صدا میزنیم.

**جلسه سی و سوم : اتصال داده های دو طرفه (**04 **:** 31**)**

میخایم زمانی که مقداری داخل اینپوت قرار میگره همزمان اون مقادیر رو داخل متغیری در vue بگیریم. برای این کار از دایرکتیو v-model استفاده میکنیم:

    <input type="email" v-model="email" required />

به این روش میگن اتصال داده دو طرفه.

**جلسه سی و چهارم : بررسی** SelectFields **(**04 **:** 36**)**

حالا به جای اینکه از اینپوت استفاده کنیم میخایم از سلکتور استفاده کنیم و مقداری که انتخاب میشه رو داخل vue داشته باشیم.

<select v-model="role">

        <option value="developer">web developer</option>

        <option value="designer">web designer</option>

    </select>

که در اینجا v-model رو به خود select دادیم و مقدارش رو داخل script میتونیم داشته باشیم.

نکته : اگر v-model رو حذف کنیم میبینم که مقدار اول به صورت پیش فرض ست شده و به این دلیل هست که متغیری که برای v-model در نظر گرفتیم role هست و مقدارش هم null هست برای اینکه مقدار پیش فرضی داشته باشه میتونیم یکی از مقادیر value رو داخل متغیر role قرار بدیم.

**جلسه سی و پنچم : بررسی** Checkboxes **(**04 **:** 41**)**

برای گرفتن مقدار اینپوتی که به صورت چک باکس هست یک راه حل استفاده از v-model هست :

<input type="checkbox" required v-model="terms"/>

و راه حل دیگری هست برای مواقعی که چندین چک باکس داریم.

<div>

        <input type="checkbox" value="ali" v-model="names">

        <label>ali</label>

        <input type="checkbox" value="nader" v-model="names">

        <label>nader</label>

        <input type="checkbox" value="bahram" v-model="names">

        <label>bahram</label>

    </div>

و یک متغیر درست میکنیم از نوع ارایه ای و به نام names که با تیک خوردن هر کدام از این اینپوت ها مقدار value به این متغیر اضافه خواهد شد.

**جلسه سی و ششم : ایونت های کیبورد و اصلاح کننده ها (**04 **:** 48**)**

میخایم در ادامه مثال جلسه قبل یک اینپوت درست کنیم که کاربر مهارت خودش رو اضافه کنه و بعد از اینکه کاما رو زد مهارت اضافه بشه و اینپوت برای اضافه کردن مهارت بعدی خالی بشه.

 <label>Skills: </label>

    <input type="text" required v-model="tempSkill" @keyup="addSkill" />

و :

addSkill(e) {

      if (e.key === "," && this.tempSkill) {

        if (!this.skills.includes(this.tempSkill)) {

          this.skills.push(this.tempSkill);

        }

        this.tempSkill = "";

      }

    },

بعد از اینکه دکمه کاما زده شد ابتدا چک میکنیم که مقداری پشت کاما وجود داشته باشه (tempSkill) بعد چک میکنیم که مقداری که وارد شده وجود نداشته باشه و در بعد اون مقدار رو به متغیر ارایه ای skills اضافه میکنیم وبعد مقدار متغیر tempSkill رو خالی میکنیم.

**جلسه سی و هفتم : چالش – حذف توانایی ها (**05 **:** 03**)**

ابتدا در جایی که مهارت ها رو داریم نمایش میدیم :

  <div v-for="skill in skills" :key="skill" class="pill" >

    <span @click="deleteSkill(skill)">{{skill}}</span>

  </div>

و بعد :

 deleteSkill(skill){

        this.skills = this.skills.filter((item) => {

            return skill !== item

        })

    }

اینجا گفتیم هر چیزی که با مقداری که در ورودی متد داریم برابر نبود رو برای من برگردون و داخل ارایه skills دوباره قرار بده.

**جلسه سی و هشتم : سابمیت کردن فرم (**05 **:** 09**)**

میخایم برای فرمی که طراحی کردیم یک دکمه طراحی کنیم برای سابمیت کردن فرم. مشکلی که وجود داره وقتی که روی دکمه submit فرم کلیک میکنیم صفحه رفرش میشه ولی ما نمیخایم این اتفاق بیفته.

به همین دلیل در تگ فرم :

<form @submit.prevent="handleSubmit">

از prevent استفاده میکنیم.

حالا میخایم اگر پسوردی که کاربر وارد میکنه کمتر از 5 کارکتر بود هنگام سابمیت کردن فرم ارور نمایش بدیم و اجازه سابمیت کردن رو ندیم:

this.passwordError = this.password.length > 5 ?

        "" : "password its not more than 5 chars";

در اینجا به جای if از if یک خطی استفاده کردیم.

**جلسه سی و نهم : علت استفاده از** vueRouter **(**05 **:** 27**)**

تا اینجا یک صفحه داشتیم که کامپوننت های مختلف رو داخلش فراخوانی میکردیم حالا فرض میخایم کلا یک صفحه دیگه ای رو داشته باشیم.

Vue router یک سیستم هست که به ما اجازه میده صفحات دیگه رو داشته باشیم و اون صفحات رو برای ما مدیریت میکنه. به عنوان مثال زمانی که ما صفحه mysite رو در url سرچ میکنیم روتر میاد میگرده دنبال صفحه ای که برای این روت وجود داره و میاد اون رو داخل صفحه مانت میکنه. و اگر صفحه جدید سرچ کنیم صفحه قبلی رو انمونت میکنه و صفحه جدید رو مانت میکنه.

Spi به این معنی هست که ما فقط یک درخواست به سرور میزنیم و یک صفحه رو درخواست میکنیم. و بعد از اون با سرور کاری نداریم. و اگر خواستیم از صفحات دیگه استفاده کنیم از سیستمی که ویو در اختیار ما قرار داده به نام vue router استفاده میکنیم تا این مدیریت پیچ ها رو بسپاریم به دست ویو.

**جلسه چهلم : نصب روتر برای پروژه (**05 **:** 33**)**

مراحل ساخت پروژه به صورت قبل هست و فقط باید router رو در هنگام ساخت فعال کنیم. در مرحله بعد از ما میپرسه که ایا برای روتر historymode میخاید داشته باشیم که yes رو میزنیم. و روند قبل رو طی میکنیم.

نکته : اگر در فایل package بریم روتر رو که نصب شده میتونیم ببینیم.

زمانی که پروژه رو باز کنیم یک فولدر جدید میبینیم به نام router که یک فایل به نام index.js داخلش هست و یک ارایه داره به نام routes که هر ابجکتی که داخلش تعریف کنیم در واقع یک روت ما هست. که این ابجکت یک سری پروپرتی داره.

پروپرتی path که همون ادرس url ای هست که بالای سایت قرار میدیم برای دسترسی به اون صفحه.

پروپرتی name که همون نامی هست که در جاهای دیگه میتونیم ازش استفاده کنیم.

پروپرتی component که صفحه ای که برای این طراحی کردیم رو اینجا ریجستر و در بالا باید ایمپورت کنیم.

حالا موقع استفاده در App.vue :

  <router-view/>

**جلسه چهل و یکم : بررسی** RouterLinks **(**05 **:** 43**)**

وقتی که پروژه را اجرا میکنیم و وارد یک روت خاص از پروژه میشویم به این صورت عمل میکند که ابتدا در فایل index.js مربوط به router به دنبال path که شبیه به url ما هست میگردد و بعد کامپوننت مربوط به آن path را در app.vue در جایی که دستور زیر نوشته شده قرار میدید:

  <router-view/>

زمانی که میخایم یک روت رو صدا بزنیم به روش های زیر میتونیم عمل کنیم:

<router-link to="/about">About</router-link> |

<a href="/about">About1</a>

تفاوت تگ router-link با a در این است که زمانی که از تگ router-link استفاده میکنیم تنها یک بار و اون هم زمانی که سایت لود میشه به سرور ریکوئست زده میشه و با زدن دکمه About دوباره به سرور ریکوئست نمیزنه. ولی اگر همین دکمه رو با تگ a بنویسیم با هر بار زدن دکمه About1 دوباره به سرور ریکوئست میزنه.

و همچنین اگر اینسپکت این المان ها رو ببینیم پروژه تگ router-link را در نهایت به تگ a تبدیل میکنه.

همچنین زمانی که از router-link استفاده میکنیم، در حالت عادی مقدار to برابر است با path صفحه که میخایم متصل بشه به About ولی برای ادرس دهی صفحه میتونیم از یک خاصیت دیگر router-link استفاده کنیم:

<router-link :to="{name:'about'}">About</router-link>

و اگر بعد path این صفحه هم تغییر کند باز هم این دکمه کار خواهد کرد.

**جلسه چهل و دوم : ساختار فولدرهای پروژه (**05 **:** 57**)**

تفاوت فولدرهای کامپوننت با ویو: توی پروژه های کوچیک معمولا به این صورت هست که داخل فولدر ویو میان پیچ ها یا همون روت ها رو قرار میدن و اگر داخل هر پیجی ما نیاز به کامپوننت داشتیم به دو نکته باید توجه کنیم :

یک : اگر اون کامپوننتی که میخایم در اون صفحه قرار بدیم مختص فقط به اون صفحه هست میریم داخل فولدر ویو یک فولدر ایجاد میکنیم مربوط به همون صفحه و اون ویو و کامپوننت های مختص به خودش رو داخلش قرار میدیم.

دو: اگر کامپوننتی که میخایم در اون صفحه قرار بدیم در جاهای دیگه هم استفاده میشه داخل فولدر کامپوننت قرار میدیم.

**جلسه چهل و سوم : بررسی پارامتر روت (**06 **:** 04**)**

Route Parameters : زمانی که در یک صفحه خاص هستیم و یک لیست داریم و میخایم با توجه به ایدی هرکدام از اجزای لیست یک کامپوننت به خصوص را صدا بزنیم، مثلا:

/jobs/123

به قسمت دوم (123) روت پارامتر میگیم و برای معرفی این کامپوننت در روتر به صورت زیر عمل میکنیم:

  {

    path: "/jobs/:id",

    name: "JobsDetails",

    component: JobsDetails,

  },

حال اگر بخایم در صفحه ای قسمت دوم روت ( روت پارامتر ) رو به دست بیاریم از دستور زیر استفاده میکنیم:

  <p>route id is:  {{ $route.params.id }}</p>

**جلسه چهل و چهارم : بررسی لینک های پویا (**06 **:** 12**)**

حال میخایم در ادامه مثال قبل در صفحه jobs روی هر کدام از متن ها که کلیک کردیم روتر خاصی به وجود بیاد.

<div v-for="job in jobs" :key="job.id" class="job">

    <router-link :to="{ name: 'JobsDetails', params : {id: job.id} }">

      <h2>{{ job.title }}</h2>

    </router-link>

  </div>

زمانی که params رو در قسمت to به این صورت معرفی میکنیم اجازه ارسال ایدی رو به کامپوننت فرزند میدیم ولی باید در روترها از props برای معرفی استفاده کنیم و بعد میتونیم ازش استفاده کنیم :

  {

    path: "/jobs/:id",

    name: "JobsDetails",

    component: JobsDetails,

    props: true

  },

**جلسه چهل و پنچم : بررسی صفحات** 404 **و** Redirects **(**06 **:** 20**)**

فرضا در مثال قبل اگر به جای روت پارامتر یک مقدار نامفهوم را وارد کنیم چیزی رو برای ما برنمیگردونه اما ما میخایم زمانیکه این اتفاق افتاد کاربر بره به صفحه 404

  {

    path: "/:catchAll(.\*)",

    name:"NotFound",

    component: NotFound

  }

و همچنین برای redirect کردن به یک صفحه :

 {

    path: "/all-jobs",

    redirect: "/jobs",

  },

حالا اگر در url، path بالا رو وارد کنیم به صفحه jobs ریدایرکت خواهیم شد.

**جلسه چهل و ششم : گذر در مسیرهای موجود** ProgrammaticNavigation **(**06 **:** 27**)**

میخایم دکمه های GoBack ، Redirect و Forward که در سمت چپ مرورگرها هست رو به صورت دستی طراحی کنیم:

<button @click="backward">Go Back</button>

  <button @click="redirect">Redirect</button>

  <button @click="forward">Forward</button>

و:

 methods: {

    backward() {

      this.$router.go(-1)

    },

    redirect() {

      this.$router.push({ name: 'home' })

    },

    forward() {

      this.$router.go(1)

    },

  },

متد go میگه در هر روتی که الان هستی با عدد منفی میتونی برگردی به روت قبل و با عدد مثبت میتونی به روت بعدی بری و اگر عدد یک باشه یک صفحه به قبل یا بعد میره و مثلا اگر 3 باشه سه صفحه به قبل یا بعد حرکت میکنه.

متد push به هر name ای که در روتر نوشتیم اشاره کنیم خواهد رفت.

**جلسه چهل و هفتم : بررسی** LazyLoding **(**06 **:** 37**)**

وقتی که پروژه رو اجرا میکنیم اگر در مرورگر کلید F12 رو بزنیم در بخش Network یک سری چیزهایی که از سرور برای ما ارسال شده میبینیم. در بین این فایل ها یک فایل به اسم app.js برای ما ارسال شده که به این فایل باندل جاوااسکریپت میگیم.

حالتی هست که تمام کامپوننت ها و ساختارهایی که داخل صفحه وجود دارد رو اینپورت میکنه و در قالب یک باندل به ما برمیگردونه. ولی ما گاهی اوقات نمیخایم تمام این کامپوننت ها و ساختارهامون رندر بشه و در قالب یک باندل برگردونده بشه. شاید بخایم هرموقع که به هر کدوم از این موارد نیاز داریم همون موقع قسمت مورد نظر رو رندر کنه. این کار باعث میشه سرعت بالا بره و از lazy loading برای این کار استفاده میکنیم.

برای این کار باید در روتر ها تغییرات زیر رو انجام بدیم:

{

    path: "/",

    name: "home",

    component: () => import("../views/HomeView.vue"),

  },

این کار را برای تمامی روت ها انجام میدیم و دیگر به قسمت اینپورت جداگانه نیازی نداریم. حال اگر قسمت network رو نگاه کنیم برای هر روتی که کلیک میکنیم همان لحظه قسمت مورد نظر رندر میشه و در قالب یک فایل باندل برگردوند میشه.

اسم این باندل ها هم میتونیم خودمون تغییر بدیم :

 {

    path: "/about",

    name: "about",

component: () => import(/\*webpackChunkName: "Home"\*/"../views/AboutView.vue"),

  },

در این جا با رفتن به روت about یک باندل برای این روت همان لحظه رندر میشه به نام Home.

**جلسه چهل و هشتم : طریقه استفاده از پروژه های اماده (**06 **:** 45**)**

زمانی که یک پروژه رو مثلا از گیت هاب دانلود میکنیم همه فولدر های پروژه به جز node\_modules وجود دارند.

به این دلیل این فایل وجود ندارد که وقتی ما یک پروژه رو میسازیم ماژول های لازم هنگام ساخت پروژه نصب میشه و اگر ما بخایم از پروژه استفاده کنیم باید به صورت دستی این پکیچ ها رو نصب کنیم:

npm install

با این دستور تمام پکیچ های مورد نیاز پروژه نصب میشه.

**جلسه چهل و نهم : استفاده از** JsonServer **(**06 **:** 48**)**

میخایم به یک سرور رکوئست بزنیم و اطلاعات رو به نمایش بزاریم.

در اینجا از ابزاری استفاده میکنیم به نام json server که به صورت لوگال برای ما یک ادرس api درست میکنه تا بتونیم به صورت دستی یک فایل جیسان درست کنیم و اطلاعات اون رو به نمایش بزاریم.

End Point : به ادرس api هایی که میخایم ازشون استفاده کنیم endpoint میگن.

نصب پکیچ json server :

npm install json-server

حالا ساخت end point برای فایل جیسانی که داریم :

npx jons-server –watch data/dbjson

بعد از کلمه watch ادرس فایل جیسانی که داریم رو قرار میدیم.

**جلسه پنچاهم : طریقه درخواست برای دریافت داده** FetchData **(**07 **:** 02**)**

 mounted() {

    fetch("http://localhost:3000/jobs")

      .then((res) => res.json())

      .then((data) => (this.jobs = data))

      .catch((err) => console.log(err.message));

  },

در اینجا از هوکس mounted استفاده کردیم تا زمانی که کامپوننت اجرا میشه دستور fetch هم اجرا بشه. در متد fetch ابتدا باید ادرس یا همون end point مربوط به api را وارد کنیم بعد فایل json رو میگیریم و داخل یک متغیر ارایه ای به نام jobs میریزیم و در نهایت گفتیم اگر با خطایی مواجه شدی متن خطا رو در کنسول نمایش بده.

**جلسه پنچاه و یکم : نمایش داده ها با درنظر گرفتن یک شرط (**07 **:** 10**)**

میتونیم از v-if برای اینکه ببینم دیتا خالی هست یا خیر استفاده کنیم.

**جلسه پنچاه و دوم : پیش نمایش و راه اندازی پروژه (**07 **:** 15**)**

پروژه ثبت، ویرایش، حذف پروژکت. و فیلتر بر اساس وضعیت پروژکت ها.

پروژه خام vue رو درست میکنیم. اطلاعات اضافی رو پاک میکنیم و پروژه رو اجرا میکنیم.

**جلسه پنچاه و سوم : بررسی صفحه اصلی و داده ها**

فایل جیسان مورد نیاز پروژه رو ایجاد میکنیم و با پکیچ jsonserver، endpoint ها رو میسازیم.

**جلسه پنچاه و چهارم : ساختار کلی پروژه در** Template **(**07 **:** 31**)**

میخایم برای project از یک کامپوننت استفاده کنیم و بعد با صدا زدن کامپوننت اونها رو به نمایش بزاریم.

**جلسه پنجاه و پنجم : استفاده از** MaterialIcons **(**07 **:** 40**)**

میخایم از ایکون هایی برای ویرایش، حذف و تغییر وضعیت استفاده کنیم. که از وب سایت material.io برای استفاده از ایکون ها استفاده میکنیم.

**جلسه پنجاه و ششم : حذف هر پروژه با کلیک (**07 **:** 47**)**

میخایم به واسطه fetch یک رکوئست به فایل جیسان بزنیم ولی این بار با role دیلیت یکی از ایجکت های project را حذف کنیم. تایپ پیش فرض برای fetch ، get هست.

methods: {

    deleteProject() {

      fetch(this.url, {method: 'DELETE'})

        .then(() => this.$emit('delete', this.project.id))

        .catch((err) => console.log(err.message));

    },

خط اول ( fetch ) عملیات حذف از فایل جیسان رو برای ما انجام میده اما فقط در فایل جیسان اون رو حذف میکنه ولی همچنان اون ابجکت رو به ما نمایش میده. برای اینکه بتونیم بعد از حذف ابجکت رو آبدیت کنیم از emit استفاده میکنیم و در کامپوننت والد ابجکت projects رو ابدیت میکنیم:

methods: {

    handleDelete(id) {

      this.projects = this.projects.filter((item) => {

        return item.id != id;

      });

    },

**جلسه پنجاه و هفتم : تکمیل هر پروژه با کلیک (**07 **:** 58**)**

میخایم وضعیت هر پروژه رو تغییر بدیم یعنی ابجکت complete رو تغییر بدیم.

اول باید برای دکمه complete یک متد درست میکنیم:

 changeComplete() {

      fetch(this.url, {

        method: "PATCH",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({ complete: !this.project.complete }),

      })

        .then(() => this.$emit("complete", this.project.id))

        .catch((err) => console.log(err.message));

    },

اول نوع متد رو به patch تغییر دادیم به این معنی که یکی از مقادیر ابجکت رو میخایم تغییر بدیم. بعد نوع فایلی که قراره بفرستیم به فایل جیسان که محتوای اون فیلدی که میخایم تغییرش بدیم داخلش هست رو مشخص میکنیم. application/json یعنی یک فایل از جنس جیسان فرستاده میشه.

بعد یک متغیر ایجاد میکنیم به نام complete و مقدارش رو برابر برعکس مقداری که الان هست قرار میدیم و ارسال میکنیم برای تغییر به فایل جیسان. بعد از این کار باید ارایه ای هم که ساختیم برای نمایش اون هم ابدیت کنیم که به وسیله emit ایدی پروژه رو ارسال میکنیم به کامپوننت والد و در کامپوننت والد :

 handleComplete(id) {

      let p = this.projects.find((item) => {

        return item.id === id;

      });

      p.complete = !p.complete;

    },

مقدار کامپوننت رو در ارایه ای که داریم آبدیت میکنیم.

**جلسه پنجاه و هشتم : ایجاد فرم پروژه های جدید (**08 **:** 18**)**

میخایم یه فرم برای ایجاد یه پروژه جدید درست کنیم.

**جلسه پنجاه و نهم : اضافه کردن پروژه ایجاد شده به اپلیکیشن (**08 **:** 26**)**

یک متد برای اضافه کردن به فایل جیسان درست میکنیم :

 handleSubmit() {

      let project = {

        title: this.title,

        details: this.details,

        complete: false,

      };

      fetch("http://localhost:3000/projects/", {

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify(project),

      }).then(() => {

        this.$router.push("/")

        .catch(err => console.log(err.message));

      });

    },

یک ابجکت با ارایه های شبیه به ابجکتی که در فایل جیسان داریم درست میکنیم بعد مقادیرش رو از اینپوت های فرم پر میکنیم و میدیم به فایل جیسان.