### Step1.

新建项目文件夹

Win + R -> CMD -> cd 文件夹地址

vue init webpack-simple phone-app

cd phone-app

cnpm install

cnpm run dev

文件夹src下新建 components

about

文件夹components 下新建 Admin.vue

Header.vue

Home.vue

Login.vue

Menu.vue

Register.vue

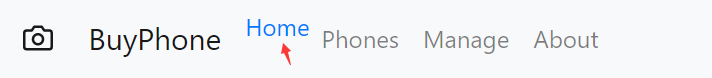
文件夹about 下新建 About.vue

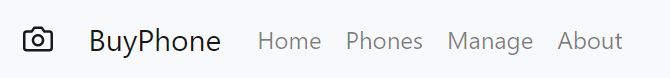
### Step2.

使用bootstrap4 CSS only

复制到index.html -> <head>

Header.vue 找个svg矢量图

class = "nav-link" 使用前

使用后

class = "ml-auto" 自动居右

驼峰写法:

components:{

    appHeader: Header1

  }

杠写法

components:{

    "app-header": Header1

  }

### Step3.

安裝路由

---> npm install vue-router -save

在main.js中引入路由

---> import VueRouter from 'vue-router'

使用路由

---> Vue.use(VueRouter)

配置路由

---> const routes = [

  {path: '/', component: Home1},

  {path: '/menu', component: Menu1},

  {path: '/admin', component: Admin1},

  {path: '/about', component: About1},

  {path: '/login', component: Login1},

  {path: '/register', component: Register1}

]

引入Home組件

---> import Home1 from './components/Home.vue'

import Menu1 from './components/Menu.vue'

import Admin1 from './components/Admin.vue'

import About1 from './components/about/About.vue'

import Login1 from './components/Login.vue'

import Register1 from './components/Register.vue'

實例化一個router, 把routes放進router中去, 再配置一個mode,如果不配置mode它會給我們加一個 “#”

---> const router = new VueRouter ({

  routes,

  mode: 'history'

})

把router放進實例中使用

---> 在new vue中的el下面加router

new Vue({

  el: '#app',

  router,

  render: h => h(App)

})

組件中Header中a標簽的href改成應該要調轉的路徑

---> <li><a href="/" class="nav-link">Home</a></li>

               <li><a href="/menu" class="nav-link">Phones</a></li>

在App.vue中添加div顯示路由跳轉后的内容

---> <div class="container">

       <router-view></router-view>

      </div>

使用router-link代替組件中Header中a標簽,目的是達到不刷新跳轉路徑,href改成屬性to

---> <li><router-link to="/" class="nav-link">Home</router-link></li>

                <!-- class="nav-link"的作用是去掉li的默认样式 -->

               <li><router-link to="/menu" class="nav-link">Phones</router-link></li>

                <li><router-link to="/admin" class="nav-link">Admin</router-link></li>

                <li><router-link to="/about" class="nav-link">About</router-link></li>

                <li><router-link to="/login" class="nav-link">Login</router-link></li>

                <li><router-link to="/register" class="nav-link">Register</router-link></li>

### Step4.路由细节功能

router-link 其实在网页中默认是 a 标签, 也可以使用 tag属性修改

---> <li><router-link **tag="div"** to="/" class="nav-link">Home</router-link></li>

router-link 中的to 可以去绑定一个动态的路由地址, 借助script中的data属性 ---> ......

<li><router-link **:to="homeLink"** class="nav-link">Home</router-link></li>

......

<script>

export default {

    data(){

        return {

            homeLink : '/'

        }

    }

}

</script>

将没有找到的路由引导到指定页面

---> 在main.js中routes中添加一个path

const routes = [

  {path: '/', component: Home1},

  {path: '/menu', component: Menu1},

  {path: '/admin', component: Admin1},

  {path: '/about', component: About1},

  {path: '/login', component: Login1},

  {path: '/register', component: Register1},

**{path: '\*', redirect: '/'}**

]

### Step5.路由name属性及跳转方法

给路由绑定name属性

---> const routes = [

  {path: '/', **name: 'homeLink'**, component: Home1},

  {path: '/menu', **name: 'menuLink'**, component: Menu1},

  {path: '/admin', **name: 'adminLink'**, component: Admin1},

  {path: '/about', **name: 'aboutLink'**, component: About1},

  {path: '/login', **name: 'loginLink'**, component: Login1},

  {path: '/register', **name: 'registerLink'**, component: Register1},

  {path: '\*', redirect: '/'}

]

--------------------------------------------------------------------------------

                <li><router-link :to="{name: 'homeLink'}" class="nav-link">Home</router-link></li>

                <li><router-link :to="{name: 'menuLink'}" class="nav-link">Phones</router-link></li>

                <li><router-link :to="{name: 'adminLink'}" class="nav-link">Admin</router-link></li>

                <li><router-link :to="{name: 'aboutLink'}" class="nav-link">About</router-link></li>

<li><router-link :to="{name: 'loginLink'}" class="nav-link">Login</router-link></li>

                <li><router-link :to="{name: 'registerLink'}" class="nav-link">Register</router-link></li>

定义方法跳转,比如定义个button进行跳转

---> 在home中添加个button

<button @click='go' class="btn btn-success">Let's order!</button>

跳转到上一次浏览的页面

--->

<script>

export default {

    methods: {

        go(){

**this.$router.go(-1)**

        }

    }

}

</script>

跳转到指定的页面

方法1

---> **this.$router.replace('/menu')**

方法2

---> **this.$router.replace({name: 'menuLink'})**

方法3(压栈, push可以传更多的参数)

---> **this.$router.push('/menu')**

方法4

---> **this.$router.push({name: 'menuLink'})**

### Step6.二级路由三级路由

在about文件下创建

Contact.vue

Delivery.vue

History.vue

OrderingGuide.vue

在about.vue中使用bootstrap布局进行调整

---> class:“row mb-5” (mb-5 🡪 margin-border 5)

class : “col-4” (分4列,展示导航)

div.list-group.mb-5

router-link tag=“li” class= “nav-link” :to=name

a.list-group-item.list-group-item-action

class : “col-8” (分8列,展示导航所对应的内容)

<router-view>

--->

<div class="row mb-5">

        <div class="col-4">

            <div class="list-group">

                <router-link tag="li" class="nav-link" :to="{name : 'historyLink'}">

                    <a class="list-group-item list-grout-item-action">历史订单</a>

                </router-link>

                <router-link tag="li" class="nav-link" :to="{name : 'contactLink'}">

                    <a class="list-group-item list-grout-item-action">联系我们</a>

                </router-link>

                <router-link tag="li" class="nav-link" :to="{name : 'orderingGuideLink'}">

                    <a class="list-group-item list-grout-item-action">手机文档</a>

                </router-link>

                <router-link tag="li" class="nav-link" :to="{name : 'deliveryLink'}">

                    <a class="list-group-item list-grout-item-action">快递信息</a>

                </router-link>

            </div>

        </div>

        <div class="col-8">

            <router-view></router-view>

        </div>

    </div>

在main.js中引入二级路由

--->

import Contact1 from "./components/about/Contact.vue"

import History1 from "./components/about/History.vue"

import Delivery1 from "./components/about/Delivery.vue"

import OrderingGuide1 from "./components/about/OrderingGuide.vue"

在routes的about中添加children属性

---> {path: '/about', name: 'aboutLink', component: About1, **children: [**

**{path: '/about/contact', name: 'contactLink', component: Contact1},**

**{path: '/about/history', name: 'historyLink', component: History1},**

**{path: '/about/delivery', name: 'deliveryLink', component: Delivery1},**

**{path: '/about/orderingGuide', name: 'orderingGuideLink', component: OrderingGuide1},**

**]**},

在about文件夹下面新建contact文件夹,contact文件夹下面新建

PhoneNumber.vue

PersonName.vue

在contact.vue中添加router-link和router-view

--->

          <router-link :to="{name: 'phoneNumberLink'}">联系电话</router-link>

            <router-link :to="{name: 'personNameLink'}">联系人</router-link>

            <router-view></router-view>

在main.js中引入三级路由

--->

import PersonName1 from "./components/about/contact/PersonName.vue"

import PhoneNumber1 from "./components/about/contact/PhoneNumber.vue"

三级路由,在routes的about中children属性里的contact中添加children属性

--->

{path: '/about/contact', name: 'contactLink', component: Contact1,**children: [**

**{path: '/about/contact/phoneNumber', name: 'phoneNumberLink', component: PhoneNumber1},**

**{path: '/about/contact/personName', name: 'personNameLink', component: PersonName1}**

**]**},

设置默认展示页面

--->

{path: '/about', name: 'aboutLink', component: About1, **redirect: '/about/contact/phoneNumber'**, children: [

…

]}

### Step7.导航守卫

**\*\*\*全局守卫**

作用: 用户没有完成登录操作的时候不可以进入系统,并通过弹窗告知用户

🡪

**router.beforeEach((to, from, next) => {**

**alert('还没有登录,请先登录!')**

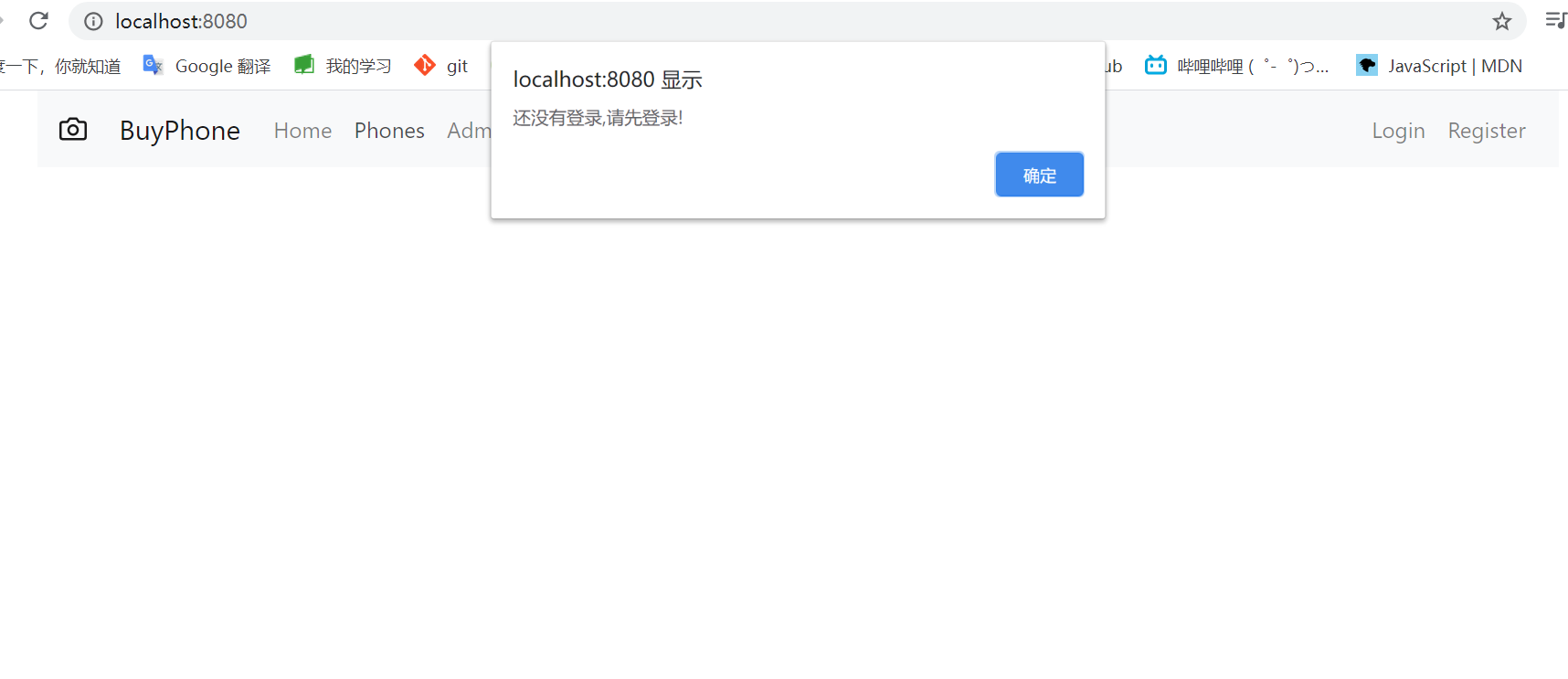
**})**

**-----------------------------------------------------------------------------------**

To 🡪 要进入哪个路由

From 🡪 从哪个路由离开

Next 🡪 对应一个函数,决定是否展示用户即将看到的页面



🡪

**router.beforeEach((to, from, next) => {**

**alert('还没有登录,请先登录!')**

**next()**

**})**



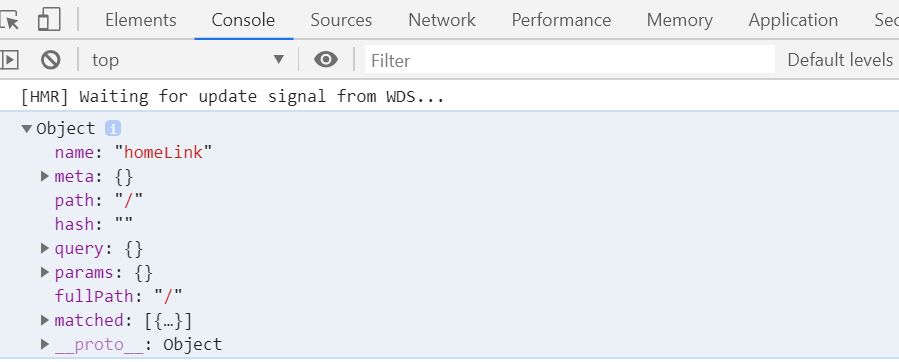
可以显示出来即将看到的页面.

🡪

**router.beforeEach((to, from, next) => {**

**console.log(to);**

**})**



🡪

**router.beforeEach((to, from, next) => {**

**if(to.path == '/login' || to.path == '/register'){**

**next();**

**}else{**

**alert('请先登录!');**

**next('/login');**

**}**

**})**

如果到Login或者Register页面的时候,正常跳转

不然就不会弹窗提醒,并被路由到Login页面中

\*\*\***全局后置钩子 (应用较少)**

🡪

router.afterEach((to, from) => {

})

就是每次跳转路由的时候都会执行一次.

\*\*\***路由独享守卫**

假设控制adminLink路由

🡪

{path: '/admin', name: 'adminLink', component: Admin1, **beforeEnter:(to, from, next) => {**

**alert('非登录状态,不能访问此页面')**

**}**}

此时无法访问admin页面,并有弹窗提示

\*\*\***组件内的守卫1**

例如在组件Admin中实现

🡪

<script>

export default {

    data(){

        return {

            name: 'Dyc'

        }

    },

}

</script>

定义一个data,返回一个name对线.

Question:为什么设置data?

Answer: 因为在组件内的守卫,是有一种特殊情况的,就是他的next会对应回调, 他是访问不到我们访问的data属性,他们的执行顺序是不一样的,这跟他们的生命周期是有关系的.

🡪在他进入我们组件的时候,让他执行一些事情

    beforeRouteEnter : (to, from, next) => {

        alert('Hello' + this.name);

        next();

    }

使用组件内的守卫时, 会被路由独享守卫影响到.



获取不到this.name, 因为data渲染之前,beforeRouteEnter已经执行了.

\*\*下面提供一个可以获取的到的方法,在next中写对应的回调

🡪

    beforeRouteEnter : (to, from, next) => {

        next(vm => {

            alert('Hello ' + vm.name)

        })

    }

此时时异步处理数据,所以可以获取到数据



\*\*\***组件内的守卫2**

🡪

    beforeRouteLeave : (to, from, next) => {

        if(confirm('确定离开吗') == true) {

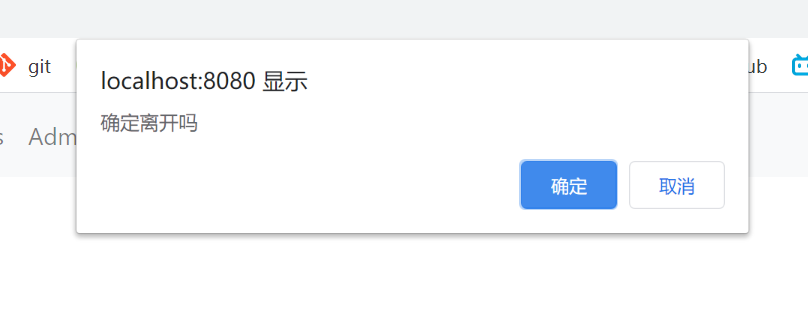
            next();

        }else{

            next(false);

        }

}



### Step8.对路由内容进行代码抽离,router-view的复用

在src中新建routes.js

把main.js中的const routes部分拷贝进去

把引入文本也拷贝进去

把routes进行公开

🡪

const routes 🡪 export const routes

在main.js中引入

🡪

import { routes } from "./routes"

\*\*\*router-view的复用

为了在每个页面中都能展示对应的消息,在App.vue中添加

🡪

<br>

    <div class="container">

      <div class="row">

        <div class="col-sm-12 col-md-4">

          <router-view name="orderingGuide"></router-view>

        </div>

        <div class="col-sm-12 col-md-4">

          <router-view name="delivery"></router-view>

        </div>

        <div class="col-sm-12 col-md-4">

          <router-view name="history"></router-view>

        </div>

      </div>

</div>

修改routes.js中的{path: '/', name: 'homeLink', component: Home1}

🡪

{path: '/', name: 'homeLink', components: {

        default: Home1,

        'orderingGuide': OrderingGuide1,

        'delivery': Delivery1,

        'history': History1

    }}

### Step9.路由的滚动行为

使用前端路由，当切换到新路由时，想要页面滚到顶部，或者是保持原先的滚动位置，就像重新加载页面那样。 vue-router 能做到，而且更好，它让你可以自定义路由切换时页面如何滚动。

**注意: 这个功能只在支持 history.pushState 的浏览器中可用。**

在Home.vue中设置

🡪

    <div  class="row">

        <div id="home" class="col-sm-12 text-center">

            <div class="background">

                <h1>欢迎大家选购手机!</h1>

                <h2>这里可能有最适合您的手机!</h2>

                <button @click='goToMenu' class="btn btn-success">Let's  order!</button>

            </div>

        </div>

    </div>

-------------------

<style>

    #home{

        background: url('../../src/assets/samsung.jpg') no-repeat center;

        height: 85vh;

        padding: 10%;

    }

    h1,h2{

        margin: 6%;

    }

    .background{

        background: #eee;

        opacity: 0.8;

        max-width: 70vw;

        margin: 0 auto;

        padding: 20px 0;

    }

</style>

Main.js里面router里添加scrollBehavior属性

🡪

const router = new VueRouter ({

  routes,

  mode: 'history',

**scrollBehavior(to, from, savedPosition){**

**return {**

**x : 0, y : 100**

**}**

**}**

})

效果: 进入Home页面的时候是在y=100的位置(假设页面纵向足够长)

🡪

return {

      selector : 'btn'

    }

效果: 进入Home页面的时候,顶端是第一个btn(假设页面纵向足够长)

🡪

    if(savedPosition){

      return savedPosition

    }else{

      return {x: 0, y: 0}

}

说明: 只有在页面中出发前进/后退按钮的时候才会有savedPosition

效果: 回到原来的滚动位置

### Step10.登录和注册结构

使用bootstrap搭建

🡪

<template>

    <div class="row mt-3">

        <div class="col-md-12 col-lg-12">

            <div class="card">

                <div class="card-body">

                    <img id="icon" class="mx-auto d-block" src="../../src/assets/galaxy-s20.png" alt="">

                    <form @submit.prevent="onSubmit">

                        <div class="form-group">

                            <label for="email">邮箱</label>

                            <input class="form-control" type="email" v-model="email">

                        </div>

                        <div class="form-group">

                            <label for="password">密码</label>

                            <input class="form-control" type="password" v-model="password">

                        </div>

                        <div class="form-group">

                            <label for="confirm-password">确认密码</label>

                            <input class="form-control" type="password

" v-model="confirmPassword">

                        </div>

                        <button type="submit" class="btn btn-block btn-success">注册</button>

                    </form>

                </div>

            </div>

        </div>

    </div>

</template>

<script>

export default {

    data(){

        return {

            email: '',

            password: '',

            confirmPassword: '',

        }

    },

    methods: {

        onSubmit(){

        }

    }

}

</script>

分析:

row --- 行

mt-3 --- 与下面有些距离

mx-auto --- 左右居中

d-block --- **Screen Size-**Visible on all

@submit.prevent --- 默认事件

### Step11.使用axios实现数据存储

使用firebase作为数据接口

安装axios

🡪

npm install axios –save

下载完成之后,就会在package.json里面会出现对应的axios

🡪

 "dependencies": {

**"axios": "^0.19.2",**

    "vue": "^2.5.11",

    "vue-router": "^3.1.6"

  }

**注**: 若不能再次npm run dev, 不妨试试 npm install

配置axios

在main.js中引入axios

🡪

import axios from 'axios'

axios.defaults.baseURL = 'https://buyphone-39f3e.firebaseio.com/'

在register中也引入axios

🡪

<script>

import axios from 'axios'

...

</script>

判断密码和确认密码是否相同,并测试一下res是啥

🡪

methods: {

        onSubmit(){

            if(this.password === this.confirmPassword){

                const formData = {

                    email: this.email,

                    password: this.password,

                    confirmPassword: this.confirmPassword

                }

                axios.post('/users.json', formData)

                    .then(res => {

                        console.log(res)

                    })

            }else{

                alert('两次密码不一致!')

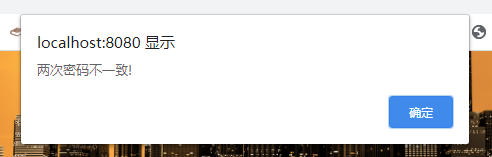
            }

        }

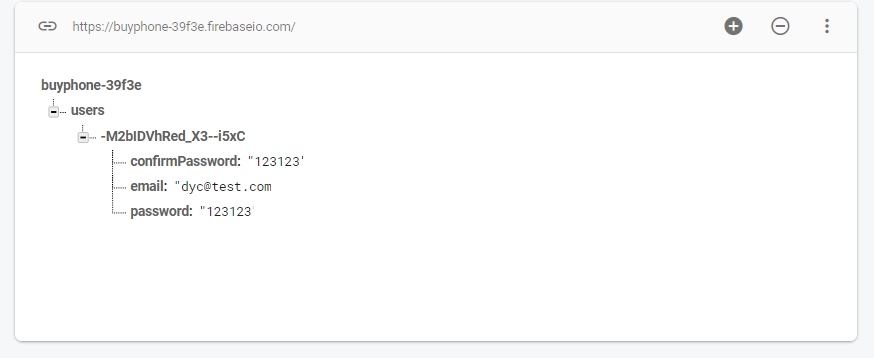
    }

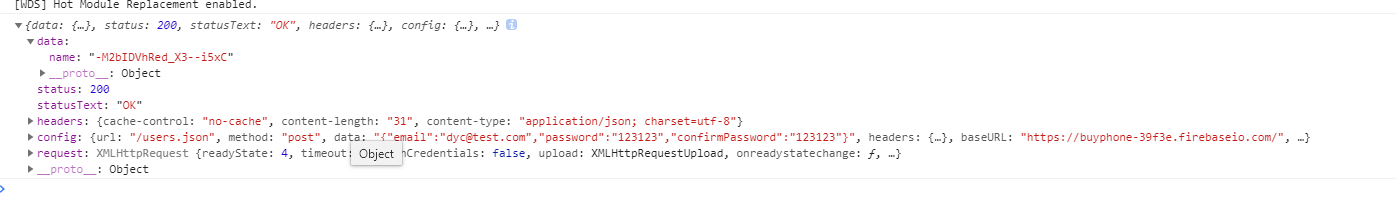
说明:

若输入不一致的两次密码,则会出现弹窗



若正确输入两次密码, 则会在firebase中生成对应的账号密码



Console.log的res会显示

完成代码,将

axios.post('/users.json', formData)

                    .then(res => {

                        console.log(res)

                    })

替换成🡪

axios.post('/users.json', formData).then(res => this.$router.push({name:'loginLink'}))

即若注册成功,则跳转登录页面

### Step12.使用axios实现登录

将拿到的对象转换成对应的数组.

在Login.vue中测试一下onSubmit中console什么

🡪

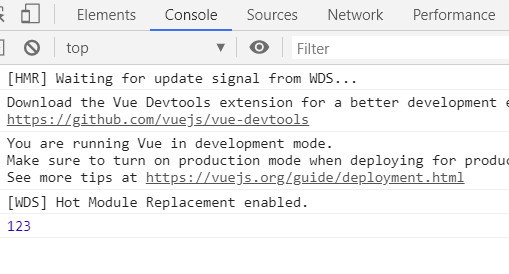
    methods: {

        onSubmit(){

            console.log(123)

        }

    }



可以打印出对应的123,说明没有问题

引入axios

🡪

<script>

import axios from 'axios';

……

</script>

使用axios.get来请求数据, 先测试一下res

🡪

    methods: {

        onSubmit(){

            axios.get('/users.json')

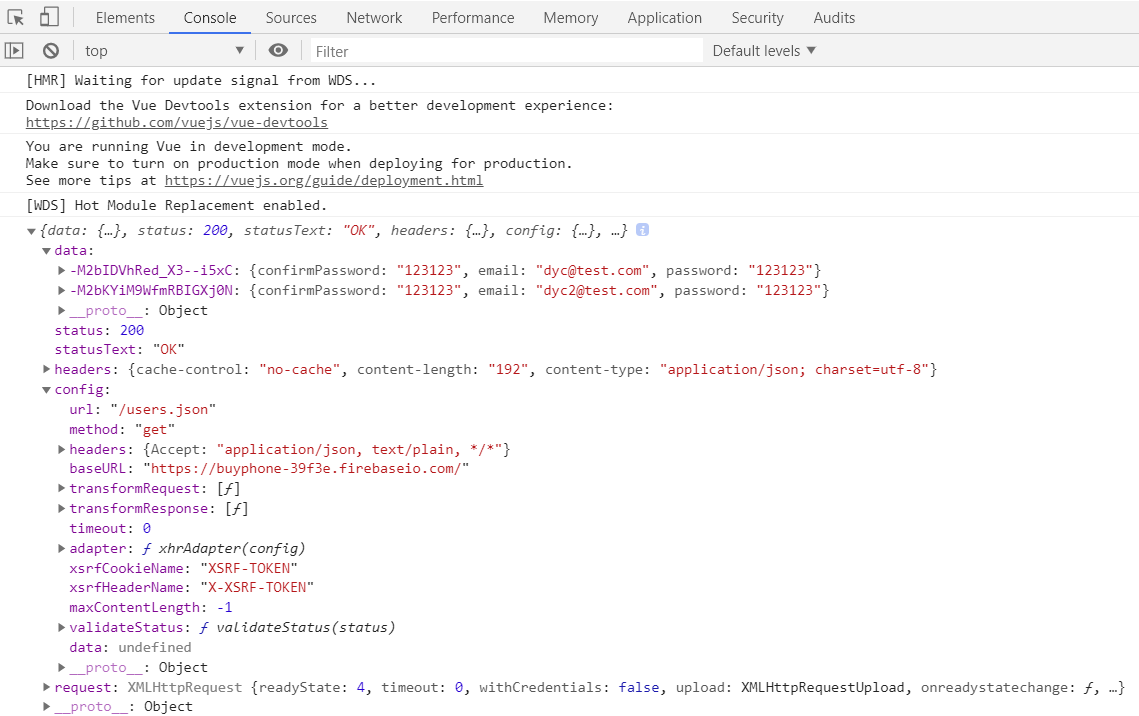
                .then(res => {

                    console.log(res)

                })

        }

    }



测试res.data

🡪

methods: {

        onSubmit(){

            axios.get('/users.json')

                .then(res => {

                    console.log(res.data)

                })

        }

    }



将这个对象转换成数组

🡪

methods: {

        onSubmit(){

            axios.get('/users.json')

                .then(res => {

                    const data = res.data;

                    const users = [];

                    for (let key in data){

                        const user = data[key]

                        users.push(user)

                    }

                    let result = users.filter((user) => {

                        return user.email === this.email && user.password === this.password;

                    })

                    if(result != null && result.length > 0){

                        this.$router.push({name: "homeLink"})

                    }else{

                        alert("账号或密码错误!")

                    }

                })

        }

    }

**说明:**

把res.data放进data中, 创建一个新数组users, 遍历data并放进数组users中, 若账号密码都对了, 就给result一个返回值, 不然返回一个空数组, 若返回不为空且长度大于1, 则跳转到Home页面, 否则弹出”账号密码错误”的警告框.

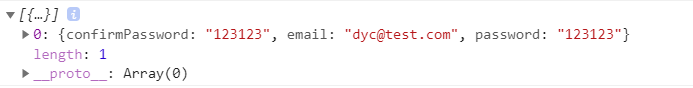
users打印出来的结果

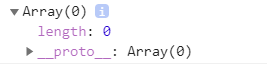
🡪



result打印出来的结果(输入正确的账号密码 / 输入错误的账号密码)

🡪





### Step13.设计购物车

# 给button添加点击事件

<td><button @click="addToBasket(item, option)" class="btn btn-sm btn-outline-success">+</button></td>

# 添加addToBasket方法

# 在data里面创建一个baskets 空数组用来装数据

addToBasket(item, option){

            this.baskets.push({

                brand: item.brand,

                size: option.size,

                price: option.price,

                quantity: 1

            })

        }

quantity 用来计数, 每次 +1

# v-if v-else

            <div v-if="baskets.length > 0">

……

            </div>

            <div v-else>

                {{basketText}}

            </div>

如果baskets里面有内容, 显示内容. 反之显示basketText

# Array.prototype.splice()

this.baskets.splice(this.baskets.indexOf(item), 1);

删除点击出的baskets数据.

# 用computed来动态获取总价:

    computed: {

        total() {

            let totalCost = 0;

            for(let index in this.baskets){

                totalCost += this.baskets[index].price \* this.baskets[index].quantity;

            }

            return totalCost;

        }

使用for…in的时候记得要let

Do 🡪 for(let index in this.baskets)

Don’t do 🡪 for(index in this.baskets)

### Step14.设计管理页面

# 设计newPhone组件,然后在admin.vue中引入

<div class="col-sm-12 col-md-8">

            <app-new-phone></app-new-phone>

        </div>

<script>

import newPhone from "./NewPhone.vue";

export default {

    components: {

        'app-new-phone': newPhone

    }

 }

</script>

# 打印console.log(this.phone);

<button @click="addItemMune" type="button" class="btn btn-success btn-block">添加</button>

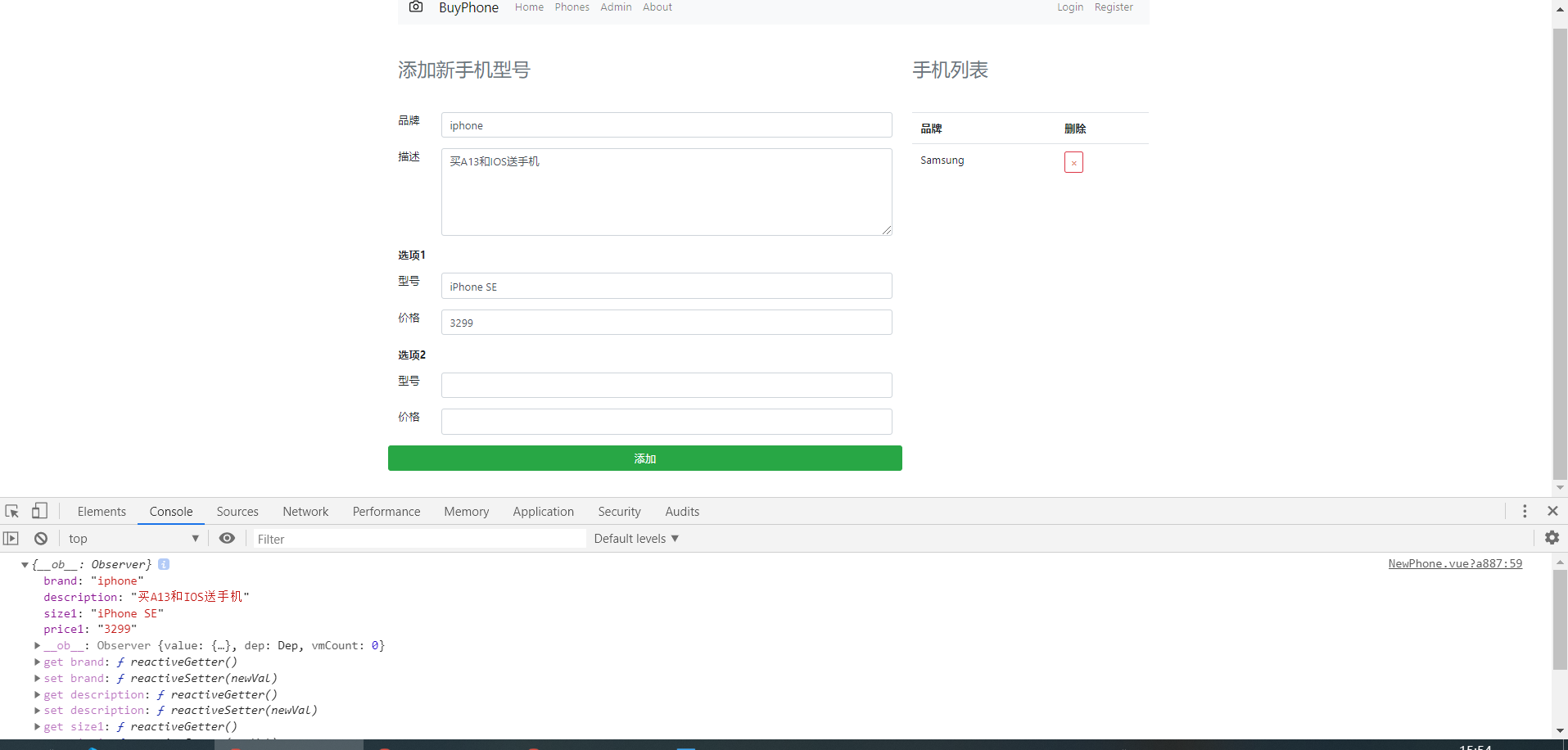
    methods: {

        addItemMune(){

            console.log(this.newPhone);

        }

    }



# 使用fetch方法将数据传到firebase

            fetch('https://buyphone-39f3e.firebaseio.com/menu.json', {

                method: 'POST',

                headers: {

                    'Content-type': 'application/json'

                },

                body: JSON.stringify(data)

            })

            .then(res => res.json())

            .then(data => this.$router.push({name: 'menuLink'}))

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

# 在admin.vue中使用fetch方法获取数据

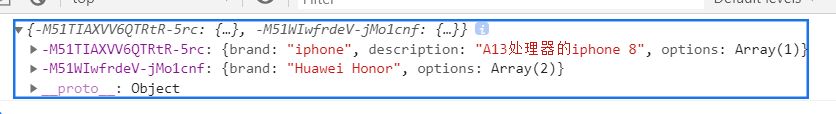
    created(){

        fetch('https://buyphone-39f3e.firebaseio.com/menu.json')

        .then(res => res.json())

        .then(data => console.log(data))

    }



# 获取object长度

    Object.keys(this.getMenuItems).length

# 获取object的第一个值

data[Object.keys(data)[0]]

# 使用created()和fetch()获取数据并渲染数据到手机列表页面

    created(){

        this.fetchData();

    },

    methods: {

        fetchData(){

            fetch('https://buyphone-39f3e.firebaseio.com/menu.json')

            .then(res => res.json())

            .then(data => {

                this.getMenuItems = data;

            })

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

# 换成使用axios获取接口数据

# 确认已经安装好了axios

# 确认已经在main.js中引入并配置好默认根路径

# 在Menu.vue中引入axios

<script>

import axios from "axios";

export default {

    ……

}

</script>

# 在fetchData函数中改用axios, 并打印一下res是什么

        fetchData(){

            // fetch('https://buyphone-39f3e.firebaseio.com/menu.json')

            // .then(res => res.json())

            // .then(data => {

            //     this.getMenuItems = data;

            // })

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

            axios.get('menu.json')

                .then(res => {console.log(res)})

        },

# 结果是:



# 然后把this.getMenuItem的值替换成get到的数据

        fetchData(){

            axios.get('menu.json')

                .then(res => {this.getMenuItems = res.data})

        }

# 简化axios的用法, 在main.js中配置vue的原型, 即 vue.prototype

// 配置Vue的原型,让axios在任何组件中不需要引入就可以使用

Vue.prototype.http = axios

# 无需引用, 直接在menu.vue中使用

        fetchData(){

            this.http.get('menu.json')

                .then(res => this.getMenuItems = res.data)

        },

# 再在NewPhone.vue中的fetch用法改成axios的用法

        addItemMune(){

            ……

            // fetch('https://buyphone-39f3e.firebaseio.com/menu.json', {

            //     method: 'POST',

            //     headers: {

            //         'Content-type': 'application/json'

            //     },

            //     body: JSON.stringify(data)

            // })

            // .then(res => res.json())

            // .then(data => this.$router.push({name: 'menuLink'}))

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

            this.http.post("menu.json", data)

                .then(res => this.$router.push({name: "menuLink"}))

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

        }

### Step15.使用vuex数据集中管理

安装vuex

* npm install vuex –save

在src下新建文件夹store

* 如果创建的文件名是index.js, 引入文件的时候直接引入路径就行了

在store文件夹中创建store.js, 然后在store.js中操作

* 引入vue和vuex

import Vue from "vue";

import Vuex from "vuex";

* 将Vuex使用一下

Vue.use(Vuex)

* 实例化一个Vuex.store的对象, 让外部能正常引入

export const store = new Vuex.store({

    state: {

        // 设置属性

    },

    getters: {

        // 获取属性的状态

    },

    mutations: {

        //  改变属性的状态

    },

    actions: {

        // 应用mutations

    }

})

* 在main.js中引入全局的store

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

* 把store在实例中应用一下

new Vue({

  el: '#app',

  router,

  store,

  render: h => h(App)

})

# 在store.js中设置一个属性

    state: {

        // 设置属性

        menuItem : {}

    }

# 在menu.vue中使用

# 设置一个setMenuItems的函数, 并把res.data传进去

        this.http.get('menu.json')

            .then(res => this.$store.commit("setMenuItems", res.data))

        },

# 这个setMenuItems是在store的mutations中的

# 参数data就是传入的res.data的形参

    mutations: {

        //  改变属性的状态

        setMenuItems(state, data){

            state.menuItem = data;

        }

    }

# 注释掉menu中data里面原来的getMenuItems, 在computed中实现getMenuItems

    computed: {

        getMenuItems(){

            // 在vuex中获取数据

        },

……

    }

# 获取vuex数据 方法1. 不安全, menuItems暴露在外面了

            // 在vuex中获取数据

            return this.$store.state.menuItems;

        }

# 方法2 使用getters

    getters: {

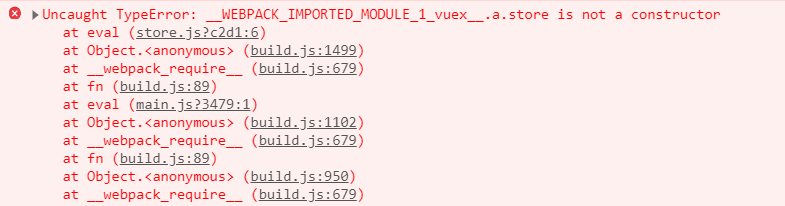
        // 获取属性的状态

        getMenuItems: state => state.menuItems

    }

# 踩坑

Uncaught TypeError: \_\_WEBPACK\_IMPORTED\_MODULE\_1\_vuex\_\_.a.store is not a constructor



原因是store.js中的

export const store = new Vuex.store({

……

})

中的没有大写成

export const store = new Vuex.Store({

……

})

# 在admin.vue中也使用, 但是此时会warn

    computed: {

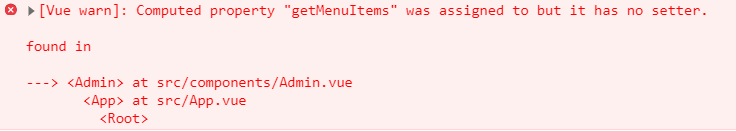
        getMenuItems(){

            return this.$store.state.menuItems;

        }

    }

[Vue warn]: Computed property "getMenuItems" was assigned to but it has no setter.



# 解决方法

getMenuItems:{

            get(){

                return this.$store.state.menuItems;

            },

            set(){

            }

        }

# 同样也可以使用getters

        getMenuItems:{

            get(){

                // return this.$store.state.menuItems;

                return this.$store.getters.getMenuItems;

            },

            set(){

            }

        }

# 在admin.vue中实现数据同步

    created(){

        fetch('https://buyphone-39f3e.firebaseio.com/menu.json')

        .then(res => res.json())

        .then(data => {

            let menuArray = [];

            for(let key in data){

                data[key].id = key;

                menuArray.push(data[key]);

            }

            // this.getMenuItems = menuArray;

            // 数据同步

            this.$store.commit("setMenuItems", menuArray)

        })

    }

# 在admin.vue中实现数据实时更新,而不是利用页面刷新而更新

    methods: {

        deleteData(item){

            fetch('https://buyphone-39f3e.firebaseio.com/menu/'+ item.id +'.json', {

                method: 'DELETE',

                headers: {

                    'Content-type' : 'application/json'

                }

            })

            .then(res => res.json())

            .then(data => {

                // 实时更新数据

                this.$store.commit('removeMenuItems', item)

            })

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

        }

    }

# 在store.js中实现removeMenuItems方法

# data就是传入的item的形参

#

    mutations: {

……

        // 将匹配到的对象,在menuItems中删除

        removeMenuItems(state, data){

            state.menuItems.forEach((item, index) => {

                if(item == data){

                    state.menuItems.splice(index, 1)

                }

            });

        }

    }

# 在NewPhone.vue中, 也实现数据实时更新

# 方法1, 直接push

methods: {

        addItemMune(){

……

            // 数据同步到vuex中

            this.http.post("menu.json", data)

                .then(res => this.$store.state.menuItems.push(data))

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

        }

    }

# 方法2, 在store.js的mutations中实现一个函数用来push数据

methods: {

        addItemMune(){

……

            // 数据同步到vuex中

            this.http.post("menu.json", data)

                .then(res => this.$store.commit('pushToMenuItems', data))

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

        }

    }

    mutations: {

        // 将新添加的phone push到menuItems中

        pushToMenuItems(state, data){

            this.state.menuItems.push(data)

        }

    }

# 设置登录后可以右上角显示邮箱的效果

# 在state和getters中都设置好当前用户和登陆状态

state: {

        // 设置属性

        menuItems : {},

**currentUser: null,**

**isLogin: false**

    },

    getters: {

        // 获取属性的状态

        getMenuItems: state => state.menuItems,

**currentUser: state => state.currentUser,**

**isLogin: state => state.isLogin**

    }

# 在header.vue中显示login和register的标签添加v-show

# 并在中间添加带有v-show的登录之后该显示的内容

                <li class="nav-link" v-show="isLogin">{{currentUser}}</li>

                <li><router-link :to="{name: 'loginLink'}" v-show="isLogin" class="nav-link">[sign out]</router-link></li>

                <li><router-link :to="{name: 'loginLink'}" v-show="!isLogin" class="nav-link">Login</router-link></li>

                <li><router-link :to="{name: 'registerLink'}" v-show="!isLogin" class="nav-link">Register</router-link></li>

# 在header.vue中导入isLogin和currentUser

    computed: {

        currentUser(){

            return this.$store.getters.currentUser;

        },

        isLogin(){

            return this.$store.getters.isLogin;

        }

    }

# 在login.vue中找到登录按钮的函数,设置登录成功之后传递currentUser和isLogin的函数

# dispatch()就是应用在actions里的

    if(result != null && result.length > 0){

        this.$store.dispatch("setUser", result[0].email);

        this.$router.push({name: "homeLink"});

    }else{

        alert("账号或密码错误!")

        this.$store.dispatch("setUser", null);

    }

# 在actions里设置setUser

    actions: {

        // 应用mutations

        setUser({commit}, user){

            commit("userStatus", user)

        }

    }

# 在mutations里面设置userStatus

# 过来的不是null, 则把过来的数据赋给state.currentUser, isLogin给true

# 过来的是null, 反之

        // 更改用户的状态信息

        userStatus(state, data){

            if(data){

                state.currentUser = data;

                state.isLogin = true;

            }else{

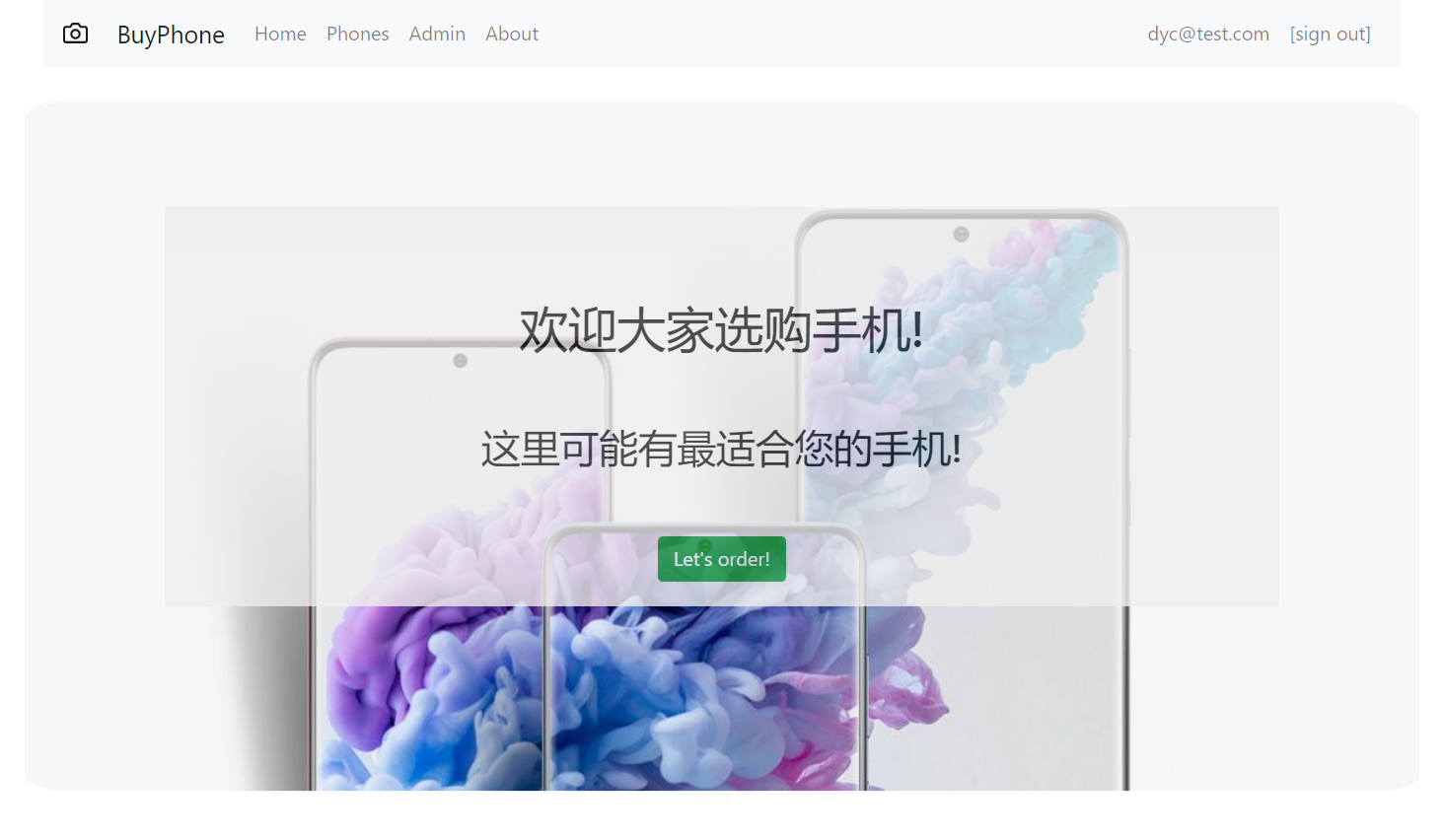
                state.isLogin = false;

                state.currentUser = null;

            }

        }

#这就是登录进去之后的样子



# 再在login.vue中设计一下导航守卫

# 像下面这样写不行! 因为识别不到this

    // 组件内的守卫

    beforeRouteEnter: (to, from, next) => {

        this.$store.dispatch("setUser", null);

    }

# 用next()回调函数

    // 组件内的守卫

    beforeRouteEnter: (to, from, next) => {

        next(vm => vm.$store.dispatch("setUser", null))

    }

# 对store文件的代码进行抽离/优化

# 抽离actions

# 在store文件夹中创建actions.js mutations.js getters.js

# 把actions里的东西剪切到actions.js里面去

        // 应用mutations

        setUser({commit}, user){

            commit("userStatus", user)

        }

# 再稍加修改 添加export 让外部可以引用它, 并让其语法正确

export const setUser = ({commit}, user) => {

    commit("userStatus", user)

}

# 删去actions里面的多余的符号, 只剩下actions就可以了

    actions

# 引入actions, \*代表全部

import \* as actions from './actions';

# 要是还需要再actions.js里面创建方法, 继续export const就可以了

export const setUser = ({commit}, user) => {

    commit("userStatus", user)

};

export const anotherFunc = () => {};

# 抽离getters

# 如上, 故技重施

// 获取属性的状态

export const getMenuItems = state => state.menuItems;

export const urrentUser = state => state.currentUser;

export const isLogin = state => state.isLogin;

# 引入

import \* as getters from './getters';

# 抽离mutations

# 故技重施

//  改变属性的状态

export const setMenuItems = (state, data) => {

    state.menuItems = data;

};

// 将匹配到的对象,在menuItems中删除

export const removeMenuItems = (state, data) => {

    state.menuItems.forEach((item, index) => {

        if(item == data){

            state.menuItems.splice(index, 1)

        }

    });

};

// 将新添加的phone push到menuItems中

export const pushToMenuItems = (state, data) => {

    state.menuItems.push(data)

};

// 更改用户的状态信息

export const userStatus = (state, data) => {

    if(data){

        state.currentUser = data;

        state.isLogin = true;

    }else{

        state.isLogin = false;

        state.currentUser = null;

    }

};

# 引入

import \* as mutations from './mutations';

# 使用module进行优化

# 在store文件夹创建module文件夹

# 创建文件(根据需要来创建, 这里我创建3个文件 menu.js users.js status.js)

# 三个js文件都放这些

const state = {

}

const getters = {

}

const mutations = {

}

const actions = {

}

export default{

    state,

    getters,

    mutations,

    actions

}

# 在menu.js里面放

const state = {

    menuItems : {}

}

# 在users.js里面放

const state = {

    currentUser: null

}

# 在status.js里面放

const state = {

    isLogin: false

}

# 再把getters里面对应的东西对应的放进去, 以menu.js为例

const getters = {

    egetMenuItems : state => state.menuItems

}

# 再把mutations里面对应的东西对应放进去, 以menu.js为例

const mutations = {

    setMenuItems (state, data){

        state.menuItems = data

    },

    removeMenuItems (state, data){

        state.menuItems.forEach((item, index) => {

            if(item == data){

                state.menuItems.splice(index, 1)

            }

        })

    },

    pushToMenuItems (state, data) {

        state.menuItems.push(data)

    }

}

# actions同理

# store.js中引入这三个文件,并使用他们

import Vue from "vue";

import Vuex from "vuex";

import menu from "./module/menu";

import users from "./module/users";

import status from "./module/status";

Vue.use(Vuex)

export const store = new Vuex.Store({

    modules: {

        menu,

        users,

        status

    }

})