Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
Факультет инфокоммуникационных технологий

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

по теме: Разработка одностраничного веб-приложения с использованием фреймворка Vue.JS
по дисциплине: Фронт-энд разработка

Специальность:
09.03.03 Мобильные и сетевые технологии

Проверил:
Добряков Д. И. _____
Дата: «__» _____ 202__г.
Оценка _____

Выполнил:
студент группы K33401
Чернов Е. К.

Санкт-Петербург 2023

ЦЕЛЬ РАБОТЫ

Получить практические навыки по работе с Vue.JS.

ВЫПОЛНЕНИЕ

Проект

По требованиям лабораторной работы требуется написать web-приложение на Vue. Я реализовал web-приложение для сети отелей основной функционал которого включает: регистрация пользователей, предоставление информации об отелей, типов номеров и самих номерах, возможность бронирования номеров и оставление комментариев для номеров.

router, stores, api

В папке router лежит файл index.js в котором прописаны пути привязаны соответствующие представления для этих путей.

index.js

```
import {createRouter, createWebHistory} from 'vue-router'
import HotelView from '../views/HotelPage.vue'
import RoomTypeView from '../views/RoomTypePage.vue'
…

const router = createRouter({
    history: createWebHistory(import.meta.env.BASE_URL),
    routes: [
        {
            path: '/',
            name: 'hotels',
            component: HotelView
        },
        {
            path: '/hotel/:id/',
            name: 'room_types',
            props: true,
            component: RoomTypeView
        },
```

```
        …
    ]
})

export default router
```

В папке stores лежат файлы (index.js, hotels.js, regCom.js, user.js), которые предоставляют функционал работы с состояниями и действиями. Разбиты файлы таким образом, что hotels.js предоставляет возможность работы с моделями базы данных Hotel, RoomType, Room нашего сервера, соответственно, regCom.js для Registration, Comment, user.js для User, Guest, Employee.

user.js

```
import {defineStore} from 'pinia'
import {userApi} from "@/api";

const useUserStore = defineStore('user', {
    state: () => ({
        users: [],
        tokens: [],
        accUser: []
    }),
    actions: {
        async loadUsers(token) {
            const response = await userApi.getUsers(token)

            this.users = response.data
            return response
        },
        …
    }
})

export default useUserStore
```

В папке api лежат файлы (index.js, instance.js, hotels.js, regCom.js, user.js), которые в свою очередь предоставляют функционал работы с API сервера. Подключение по API реализовано с помощью библиотеки Axios. В файле instance.js, помимо инициализации axios, мы добавляем interceptors для отслеживания ошибок. В нашем случае для обновления access token.

instance.js

```
import axios from "axios";

const apiURL = 'http://127.0.0.1:8000'

const instance = axios.create({
    baseURL: apiURL
})

…

instance.interceptors.response.use(function (response) {
    return response;
}, async function (error) {
    if (error.response.status === 401) {
        const originalRequest = error.config
        const refreshToken = localStorage.getItem('refreshToken')
        const response = async (refreshToken) => {
            return await instance({
                method: 'POST',
                url: `/auth/token/refresh/`,
                data: {
                    refresh: refreshToken
                }
            })
        }

        const accessToken = await response(refreshToken)
        localStorage.setItem('accessToken', accessToken.data.access)

        originalRequest.headers['Authorization'] = 'Bearer ' +
accessToken.data.access
```

```
        return instance(originalRequest)
    }
    return Promise.reject(error);
})

export default instance
```

user.js

```
class UserApi {
    constructor(instance) {
        this.API = instance
    }

    login = async (username, password) => {
        return this.API({
            method: 'POST',
            url: `/auth/token/`,
            data: {
                username: username,
                password: password
            }
        })
    }

    …

    putAccUser = async (token, id, username, firstName, lastName, email,
phoneGuest, passportGuest, phoneEmployee, positionEmployee) => {
        return this.API({
            method: 'PUT',
            url: `/account/user/${id}/`,
            headers: {'Authorization': 'Bearer ' + token},
            data: {
                username: username,
                first_name: firstName,
                last_name: lastName,
                email: email,
                user_guest: {
                    phone_guest: phoneGuest,
                    passport_guest: passportGuest
                },
```

```
            user_employee: {
                phone_employee: phoneEmployee,
                position_employee: positionEmployee
            }
        }
    })
}

deleteUser = async (token, id, currentPassword) => {
    return this.API({
        method: 'DELETE',
        url: `/auth/users/${id}/`,
        headers: {'Authorization': 'Bearer ' + token},
        data: {
            current_password: currentPassword
        }
    })
}

createUser = async (username, email, password) => {
    return this.API({
        method: 'POST',
        url: '/auth/users/',
        data: {
            username: username,
            email: email,
            password: password
        }
    })
}
}

export default UserApi
```

layout, views, components

В layout мы определяем базовый шаблон и прописываем tag <slot/> для того, чтобы на это место подставлять наши views.

BaseLayout.vue

```
<template>
    <main class="background">
      <div class="blur">
        <slot />
      </div>
    </main>
</template>

<script>
export default {
  name: "BaseLayout"
}
</script>

<style scoped>
…
</style>
```

Во views мы реализовывали интерфейсы web-приложения. В ходе работы логика реализации функционала каждого представления повторялась. Разберем основные моменты.

Передача данных от родительское объекта дочернему:

Для того, чтобы передать данные дочернему компоненту мы определяем дочерний компонент (HotelItem.vue) и с помощью директивы v-bind прописываем имена props на стороне дочернего элемента.

HotelPage.vue

```
<template>
```

```
  <base-layout>
    <nav-bar />
    <div id="hotelPage">
      <div class="container col-8 py-4">
        <h1 class="text-center">Our hotels</h1>
        <ul class="navbar-nav p-3">
          <li class="nav-item" v-for="hotel in hotels" :key="hotel.id">
            <hotel-item class="mx-0 px-0" :name_hotel="hotel.name_hotel"
:address_hotel="hotel.address_hotel" :des_hotel="hotel.des_hotel" />
            <p class="d-flex justify-content-center" id="lookButton">
              <RouterLink class="nav-link btn-text w-100 text-center fs-5"
:to="{name: 'room_types', params: {id: hotel.id}}">Look</RouterLink>
            </p>
          </li>
        </ul>
      </div>
    </div>
  </base-layout>
</template>

<script>
import …

export default {
  name: "HotelPage",
  components: { BaseLayout, HotelItem, NavBar },
  computed: {
    ...mapState(useHotelsStore, ['hotels'])
  },
  methods: {
    ...mapActions(useHotelsStore, ['loadHotels'])
  },
  mounted() {
    this.loadHotels()
  }
}
</script>

<style scoped>
…
</style>
```

Дочерний компонент на своей стороне принимает props, записывает их и отображает.

HotelItem.vue

```
<template>
  <div class="row" id="hotelItem">
    <div class="col-6">
      <p class="fs-2">{{ name_hotel }}</p>
      <p class="fs-4">Address: <span class="fs-5">{{ address_hotel
}}</span></p>
    </div>
    <div class="col-6">
      <p class="fs-2">Description:</p>
      <p v-if="des_hotel" class="fs-5">{{ des_hotel }}</p>
      <p v-else class="fs-5">We don't have any descriptions...</p>
    </div>
  </div>
</template>

<script>
export default {
  name: "HotelItem",
  props: {
    name_hotel: {
      type: String,
      required: true
    },
    address_hotel: {
      type: String,
      required: true
    },
    des_hotel: {
      type: String,
      required: false
    }
  }
}
</script>
<style scoped>
…
</style>
```

Передача данных от дочернего объекта родительскому:

Для того, чтобы передать данные от дочернего объекта в родительский мы в дочернем элементе прописываем метод, который будет отправлять данные наверх, а родительский объект будет прослушивать изменения от дочернего.

Алгоритм реализации следующий:

1. Реализуем метод, который отправляет данные наверх, используя конструкцию this.$emit;
2. В родительском объекте с помощью директивы v-on установить прослушивание;
3. Реализовать метод, который будет обрабатывать пришедшие данные.

UpdateRegForm.vue

```
<template>
  …
</template>

<script>
import …

export default {
  name: "UpdateRegForm",
  data() {
    return {
      isValid: true,
      idReg: "",
      idHotel: "",
      nameHotel: "",
      idRoomType: "",
      roomType: "",
      idRoom: "",
      numberRoom: "",
      idEmployee: "",
      statusReg: "",
      statusPay: "",
      checkIn: "",
```

```
    checkOut: "",
    isStaff: localStorage.getItem('isStaff') === "true"
  }
},
computed: {
  ...mapState(useRegComStore, ['regs']),
  ...mapState(useHotelsStore, ['hotels', 'rooms'])
},
methods: {
  ...mapActions(useRegComStore, ['loadRegs', 'delReg']),
  ...mapActions(useHotelsStore, ['loadHotels', 'loadRooms']),
  async saveBook() {
    await this.loadHotels()
    for (let hotel of this.hotels) {
      if (this.nameHotel === hotel.name_hotel) {
        this.idHotel = hotel.id
        for (let roomType of hotel.hotel_room_type) {
          if (this.roomType === roomType.type_rt) {
            this.idRoomType = roomType.id
            break
          }
          this.idRoomType = 0
        }
        break
      }
      this.idHotel = 0
    }
    await this.loadRooms(this.idRoomType)
    for (let room of this.rooms.rt_room) {
      if (this.numberRoom.toString() === room.number_room.toString()) {
        this.idRoom = room.id
        break
      }
      this.idRoom = 0
    }
    if (this.idHotel === 0 || this.idRoomType === 0 || this.idRoom === 0) {
      this.isValid = false
      return
    }
    this.isValid = true
    const stReg = (this.statusReg === "Booked") ? "B" : "T"
    const stPay = (this.statusPay === "Not paid for") ? "NP" : "YP"
```

```
      this.$emit('saveBook', {
        id_hotel: this.idHotel,
        name_hotel: this.nameHotel,
        id_rt: this.idRoomType,
        rt: this.roomType,
        id_room: this.idRoom,
        number_room: this.numberRoom,
        id_employee: this.idEmployee,
        status_reg: stReg,
        status_pay: stPay,
        check_in: this.checkIn,
        check_out: this.checkOut
      })
    },
    …
  }
}
</script>

<style scoped>
…
</style>
```

UpdateRegPage.vue

```
<template>
  <base-layout>
    <nav-bar />
    <div class="container col-5 py-4" id="updateRegPage">
      <h1 class="text-center">Change registration</h1>
      <update-reg-form @saveBook='onSaveBook' />
    </div>
  </base-layout>
</template>

<script>
import …

export default {
  name: "UpdateRegPage",
  components: { NavBar, UpdateRegForm, BaseLayout },
  computed: {
```

```
    ...mapState(useRegComStore, ['regs'])
  },
  methods: {
    ...mapActions(useRegComStore, ['updateReg']),
    async onSaveBook(data) {
      const accessToken = localStorage.getItem('accessToken')
      const idReg = this.$route.params['id']
      const idUser = localStorage.getItem('idUser')
      const booking = new Date().toJSON().slice(0, 10)
      await this.updateReg(accessToken, idReg, idUser, data.id_hotel,
data.id_rt, data.id_room, data.id_employee, data.status_reg, data.status_pay,
data.check_in, data.check_out, booking)
    }
  }
}
</script>

<style scoped>
…
</style>
```

В данном примере, мы заполняем данные в форме нажимаем на кнопку Save, данные из компонента отправляются наверх и в представлении отправляются на сервер.

ВЫВОД

В результате работы получил практические навыки по работе с Vue.JS.