САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №3

Выполнил:

Таначев Егор

Группа K33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Мигрировать ранее написанный сайт на фреймворк Vue.JS. Обязательным условием является подключение роутера, реализация работы с внешним API, разумное деление на компоненты.

Ход работы

В нашем проекте будет 3 страницы: главная, каталог и личный кабинет. Авторизация, регистрация и раздел избранные представлены в виде модальных окон, к которым можно подключиться с любой страницы. Скриншот с настройкой роутера, а также app.vue представлены на Рисунках 1-2.

```
6   // массив с роутами
7   routes: [
8     // отдельный роут
9     {
10      path: '/',
11      name: 'index',
12      component: () => import('../views/MainPage.vue')
13    },
14    {
15      path: '/catalog',
16      name: 'catalog',
17      component: () => import('../views/CatalogPage.vue')
18    },
19    {
20      path: '/account',
21      name: 'account',
22      component: () => import('../views/AccountPage.vue')
23    }
24  ]
25 })
```

Рисунок 1 – Файл router/index.js

```vue
<template>

  <!-- HEADER -->
  <HeaderPage />
  <RouterView />
  <!-- FOOTER -->
  <FooterPage />

  <!-- MODALS -->
  <ModalAccountPage />
  <ModalRegistrationPage />
  <ModalFavoritesPage />

</template>

<script>
  import HeaderPage from '@/views/HeaderPage.vue';
  import FooterPage from '@/views/FooterPage.vue';
  import ModalAccountPage from '@/views/ModalAccountPage.vue';
  import ModalRegistrationPage from '@/views/ModalRegistrationPage.vue';
  import ModalFavoritesPage from '@/views/ModalFavoritesPage.vue';
```

Рисунок 2 – App.vue

У веб-приложения есть четкая структура, по которой разбита работа с памятью (stores), серверными скриптами (api), страницы (views), оболочки (layots) и компоненты (components). Даже css-файл разбит на каждую логическую секцию сайта и подключается с помощью import. Структура проекта и пример main.css представлены на Рисунках 3-4.

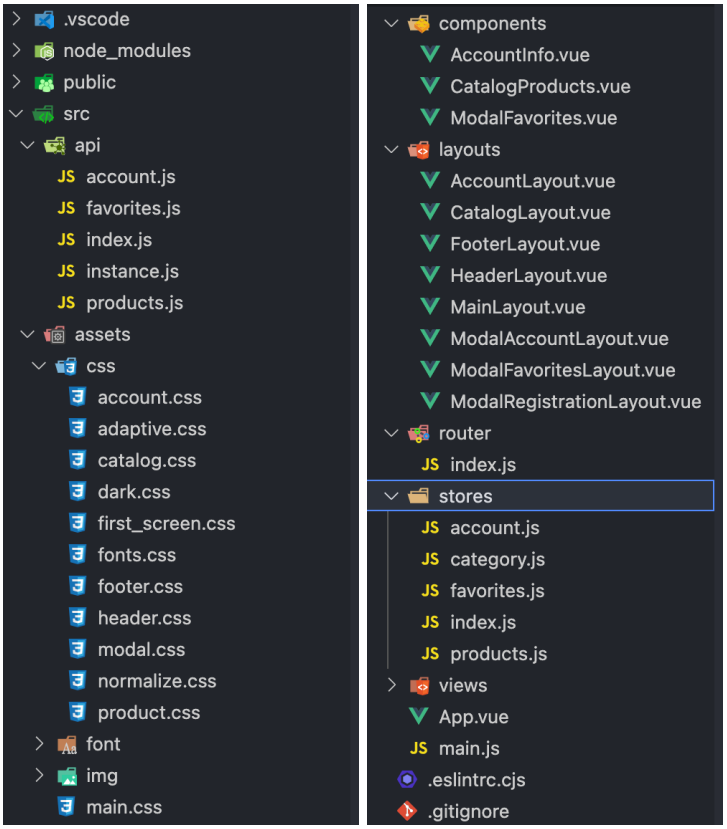

Рисунок 3 – Структура проекта

```
/* FONTS */
@import '@/assets/css/fonts.css';
/* NORMALIZE */
@import '@/assets/css/normalize.css';
/* HEADER */
@import '@/assets/css/header.css';
/* FIRST SCREEN */
@import '@/assets/css/first_screen.css';
/* ACCOUNT */
@import '@/assets/css/account.css';
/* CATALOG */
@import '@/assets/css/catalog.css';
/* PRODUCT */
@import '@/assets/css/product.css';
/* FOOTER */
@import '@/assets/css/footer.css';
/* MODAL */
@import '@/assets/css/modal.css';
/* ADAPTIVE */
@import '@/assets/css/adaptive.css';

/* DARK MODE */
@import '@/assets/css/dark.css';
```

Рисунок 4 – main.css

В этом проекте за основу работы с api я взял json server, а также подключил внешний api для Яндекс Карты в личном кабинетете, для которого кстати нужно было специально получить api-ключ. Вся работа с базами данный в проекте устроена за счет того, что мы отправляем различные CRUD-запросы и сервер в ответ нам может выдать отфильтрованные данные, создавать новые товары или даже авторизовываться с помощью accessToken и хэшировать пароль. На Рисунках 5-7 представлены основные методы.

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

    // Авторизация
    login = async (data) => {
        return this.API({
          method: 'POST',
          url: '/login',
          data,
          headers: {
            'Content-Type': 'application/json'
          }
        })
    }

    // Регистрация
    signup = async (data) => {
        return this.API({
          method: 'POST',
          url: '/signup',
          data,
          headers: {
            'Content-Type': 'application/json'
          }
        })
    }


}

export default AccountApi
```

Рисунок 5 – Авторизация и регистрация

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

    // Метод для получения всех товаров
    getAll = async () => {
      return this.API({
        url: '/products'
      })
    }

    // Метод для получения товаров по категории
    getByCategory = async (category) => {
      return this.API({
        url: `/products?category=${category}`
      })
    }
```

Рисунок 6 – Вывод товаров всех и по категориям

```
    // Получение избранных товаров
    getAll = async (userId) => {
      return this.API({
        url: `/favorites?userId=${userId}`
      })
    }

    // Удаление избранного товара
    delete = async (id, token) => {
      return this.API({
        method: 'DELETE',
        url: `/favorites/${id}`,
        headers: {
          'Authorization': `Bearer ${token}`,
          'Content-Type': 'application/json'
        }
      })
    }

    // Добавление избранного товара
    add = async (data, token) => {
      return this.API({
        method: 'POST',
        url: `/favorites/`,
        data,
        headers: {
```

Рисунок 7 – Работа с избранными товарами

Для управления состоянием мы используем pinia, где хранится вся информация о пользователе, а также сохраняются данные при отправке различных запросах. Скриншоты представлены на Рисунках 8-11.

Рисунок 8 – AccountStore

```
// создаём хранилище
const useFavoritesStore = defineStore('favorites', {
 // в стейте заведём пустой массив с товарами
 state: () => ({
    favorites: []
 }),


 actions: {
  // заведём метод для подгрузки избранных товаров
  async loadFavorites(userId) {
    const response = await favoritesApi.getAll(userId)
    this.favorites = response.data
    return response
  },

  async DeleteFavorite(id, token) {
   const response = await favoritesApi.delete(id, token)
   this.favorites = response.data
   return response
  },

  async AddFavorite(data, token) {
    const response = await favoritesApi.add(data, token)
    this.favorites = response.data
    return response
  },
```

Рисунок 9 – useFavoriteStore

```
// создаём хранилище
const useProductsStore = defineStore('products', {
 // в стейте заведём пустой массив с товарами
 state: () => ({
   products: []
 }),


 actions: {
   // заведём метод для подгрузки товаров
   async loadProducts() {
     const response = await productsApi.getAll()
     this.products = response.data
     return response
   },

   // заведём метод для подгрузки товаров
   async loadProductsCategory(category) {
    const response = await productsApi.getByCategory(category)
    this.products = response.data
    return response
  },
```

Рисунок 10 – useProductStore

Также на Рисуке 11 и 12 представлена страница cataloge. Именно сюда подгружаются и генерируются карточки из нашей бары данных.

```
        <div class="section__catalog-cards" id="catalog">
            <div
                class="section__catalog-card"
                    v-for="product in products"
                    :key="product.id"
                >
                <catalog-products
                    :id="product.id"
                    :title="product.title"
                    :price="product.price"
                    :image="product.image"
                    :category="product.category"
                    @add-favorite="addFavorite"
                >
                </catalog-products>
            </div>
        </div>
    </div>
</catalog-layout>
```

Рисунок 11 – Секция с разметкой

```vue
<script>
    import { mapActions, mapState } from 'pinia'  4.3k (gzipped: 2k)

    import useProductsStore from '@/stores/products'
    import useFavoritesStore from '@/stores/favorites'
    import { useCategoryStore } from '@/stores/category'

    import CatalogLayout from '@/layouts/CatalogLayout.vue'
    import CatalogProducts from '@/components/CatalogProducts.vue'

    export default {
        name: 'CatalogPage',

        components: { CatalogLayout, CatalogProducts },

        computed: {
            ...mapState(useProductsStore, ['products']),

            selectedCategory() {
                return useCategoryStore().selectedCategory
            },

            productsCount() {
                const count = this.products.length;
                if (count === 1) {
                    return 'товар';
                } else if (count > 1 && count < 5) {
                    return 'товара';
                } else {
                    return 'товаров';
                }
            },
        },
```

```
108
109         methods: {
110             ...mapActions(useProductsStore, ['loadProducts', 'loadProducts
111
112             async LoadProducts() {
113                 if (useCategoryStore().selectedCategory === 'Все товары')
114                     this.loadProducts()
115                 } else {
116                     this.loadProductsCategory(useCategoryStore().selectedC
117                 }
118             },
119
120             ...mapActions(useFavoritesStore, ['AddFavorite']),
121
122             async addFavorite(data) {
123                 const storageData = localStorage.getItem('pinia_accounts')
124                 const storageObject = JSON.parse(storageData);
125                 const token = storageObject.accounts.accessToken;
126
127                 const response = await this.AddFavorite(data, token)
128
129                 if (response.status === 200) {
130                     this.LoadFavorites()
131                 }
132             },
133         },
134
135         watch: {
136             selectedCategory: 'LoadProducts',
137         },
```

Рисунок 12 – Секция со скриптами

Вывод

В результате лабораторной работы нам была поставлена задача миграции ранее созданного веб-сайта на фреймворк Vue.js. Основными требованиями были подключение роутера, интеграция внешнего API, а также разумное разделение проекта на компоненты.

Для выполнения задачи были созданы три основные страницы: главная, каталог и личный кабинет. Важно отметить, что авторизация, регистрация и раздел "Избранные" были реализованы в виде модальных окон, которые можно открывать с любой страницы. Роутер был настроен для обеспечения навигации между этими страницами.

Структура проекта была четко определена, разделяя работу с данными, серверными скриптами, страницами, оболочками и компонентами. Даже файлы CSS были разделены на логические секции сайта и импортированы соответственно.