САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа 3

Выполнил:

Котовщиков Андрей

Группа K3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Мигрировать ранее разработанное приложение (в рамках ЛР1 и ЛР2) на фреймворк Vue.JS:

1. должен быть подключён роутер;
2. должна быть реализована работа с внешним API;
3. разумное деление на компоненты;
4. использование composable для выделения повторяющегося функционала в отдельные файлы.

Ход работы

Подключение роутера

Для навигации между view (страницами) приложения был подключен и настроен роутер. На рисунке 1 можно видеть список всех маршрутов.

```
const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: "/",
      name: "restaurant-list",
      component: () => import("../pages/RestaurantListPage.vue"),
      meta: { requiresAuth: true },
    },
    {
      path: "/restaurant/:id",
      name: "restaurant",
      component: () => import("../pages/RestaurantPage.vue"),
      props: true,
      meta: { requiresAuth: true },
    },
    {
      path: "/sign-in",
      name: "sign-in",
      component: () => import("../pages/SignInPage.vue"),
    },
    {
      path: "/sign-up",
      name: "sign-up",
      component: () => import("../pages/SignUpPage.vue"),
    },
    {
      path: "/profile",
      name: "profile",
      component: () => import("../pages/ProfilePage.vue"),
      meta: { requiresAuth: true },
    },
  ],
});
```

Рисунок 1 — Секция «scripts» из package.json

Для авторизации пользователей при переходе между страницами был введен специальный мета параметр «requiresAuth». Наличие и значение этого параметра проверяется при навигации. Если он установлен в true, производится проверка токена пользователя (рисунок 2).

```
router.beforeEach(async (to, from, next) => {
  const authStore = useAuthStore();
  if (to.meta.requiresAuth && !authStore.isAuthenticated) {
    return next({ name: "sign-in" });
  }

  next();
});

export default router;
```

Рисунок 2 — Проверка авторизации при навигации

Работа с внешним API

Работа с внешним API производится при помощи AXIOS. Были созданы два класса для отправки http запросов на json-server (рисунок 3).

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

  async filter(data) {
    return this.API({
      url: "/restaurants",
      params: data,
    });
  }

  async getMenus(data) {
    return this.API({
      url: "/menus",
      params: data,
    });
  }
}

export default RestaurantsApi;
```

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

  async register(data) {
    return this.API({
      method: "POST",
      url: "/register",
      data,
      headers: {
        "Content-Type": "application/json",
      },
    });
  }

  async login(data) {
    return this.API({
      method: "POST",
      url: "/login",
      data,
      headers: {
        "Content-Type": "application/json",
      },
    });
  }

  async getProfile({ accessToken }) {
    return this.API({
      url: "/users/1",
      headers: {
```

Рисунок 3 — Классы для работы с внешним api

Для декомпозиции и переиспользования UI логики и элементов все страницы (views) были поделены на компоненты. Код компонентов «Navbar» и «RestaurantCard» изображены на рисунках 4 и 5 соответственно.

```vue
<script>
export default {
  name: "Navbar",

  methods: {
    logout() {
      localStorage.clear();
      this.$router.push({ name: "sign-in" });
    },
  },
};
</script>

<template>
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-5">
    <div class="container">
      <router-link class="navbar-brand" :to="{ name: 'restaurant-list' }">Рестораны</router-link>
      <div class="navbar-nav">
        <router-link class="nav-link" :to="{ name: 'profile' }">Профиль</router-link>
        <a class="nav-link" href="#" @click.prevent="logout">Выйти</a>
      </div>
    </div>
  </nav>
</template>
```

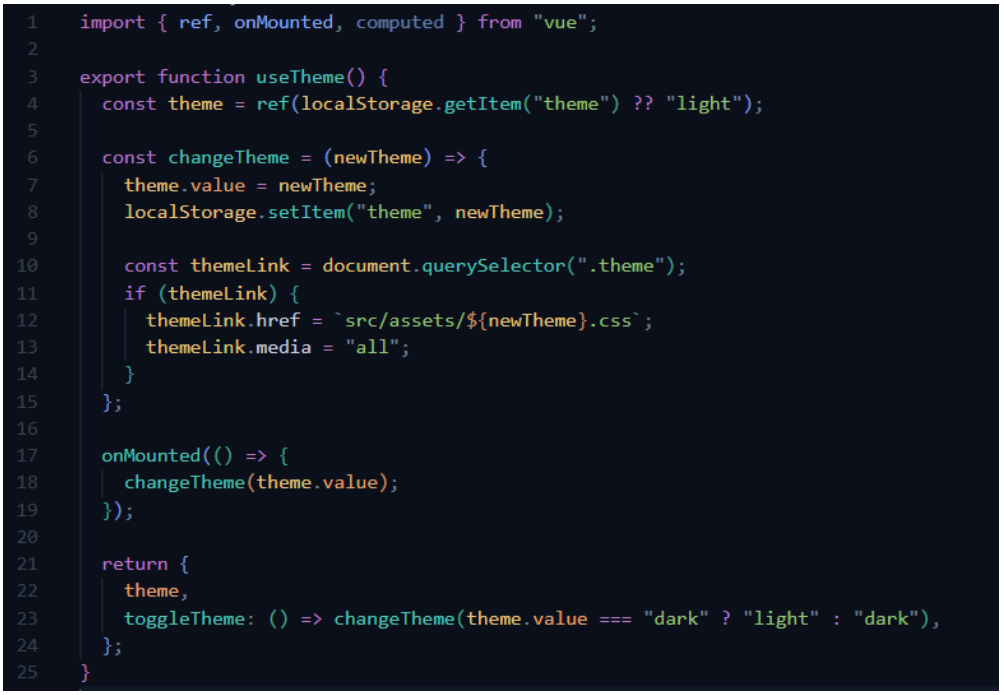
Рисунок 4 — Компонент «Navbar»

```vue
    ],
    priceFrom: {
      type: Number,
      required: true,
    },
  },
  data() {
    let cuisinesString = this.cuisines.join(", ");
    if (cuisinesString.length > 15) {
      cuisinesString = cuisinesString.substring(0, 15) + "...";
    }

    return {
      cuisinesString,
    };
  },
};
</script>

<template>
  <div class="col">
    <div class="card-bg card">
      <img
        src="https://zefirka.club/uploads/posts/2023-01/1673355119_22-zefirka-club-p-znak-voprosa-na-chernom-fone-46.jpg"
        ,
        class="restaurant-image card-img-top"
        :alt="name"
      />
      <div class="card-body">
        <h3 class="restaurant-card-title card-title">{{ name }}</h3>
        <p class="card-text">{{ description }}</p>
        <ul class="list-group list-group-flush mb-3">
          <li class="card-bg list-group-item">Город: {{ city }}</li>
          <li class="card-bg list-group-item">Кухня: {{ cuisinesString }}</li>
          <li class="card-bg list-group-item">Цена: От {{ priceFrom }} рублей</li>
        </ul>
        <router-link
          :to="{ name: 'restaurant', params: { id, name } }"
          class="btn btn-primary text-white"
          >Подробнее</router-link
```

Рисунок 5 — Компонент «RestaurantCard»

Использование composable

При помощи composable был реализован кастомный хук «useTheme» для смены темы в приложении. Код хука можно наблюдать на рисунке 6.



```
import { ref, onMounted, computed } from "vue";

export function useTheme() {
  const theme = ref(localStorage.getItem("theme") ?? "light");

  const changeTheme = (newTheme) => {
    theme.value = newTheme;
    localStorage.setItem("theme", newTheme);

    const themeLink = document.querySelector(".theme");
    if (themeLink) {
      themeLink.href = `src/assets/${newTheme}.css`;
      themeLink.media = "all";
    }
  };

  onMounted(() => {
    changeTheme(theme.value);
  });

  return {
    theme,
    toggleTheme: () => changeTheme(theme.value === "dark" ? "light" : "dark"),
  };
}
```

Рисунок 6 — Хук «useTheme»

Вывод

В ходе выполнения лабораторной работы 3 был изучен фреймворк Vue JS. Многостраничный сайт, разработанный в ходе предыдущих лабораторных работ, был полностью переписан под SPA с использованием компонентного подхода.