САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа

Выполнил:

Андронов Денис

Группа

K3342

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

Лабораторная работа 3: Разработка одностраничного веб-приложения (SPA) с использованием фреймворка Vue.JS

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

Каким требованиям ваше приложение должно соответствовать:

Должен быть подключён роутер

Должна быть реализована работа с внешним API (желательно посредством axios)

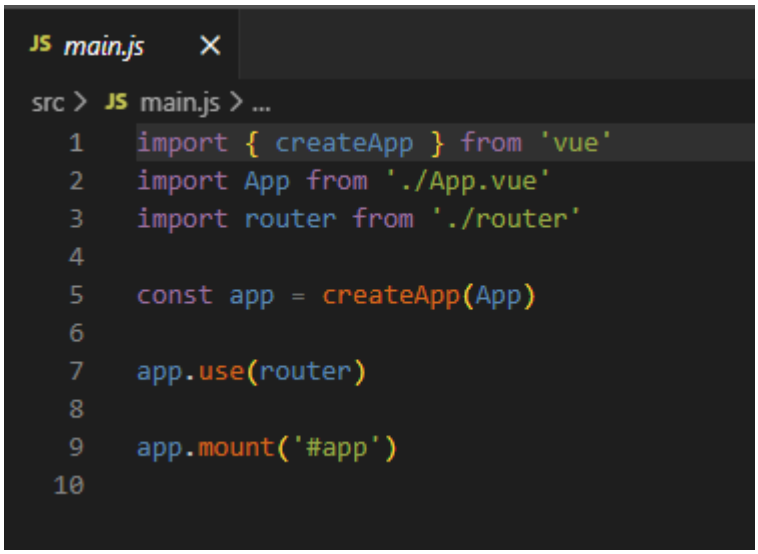
Разумное деление на компоненты (продемонстрируйте понимание компонентного подхода)

Использование composable для выделения повторяющеся функционала в отдельные файлы

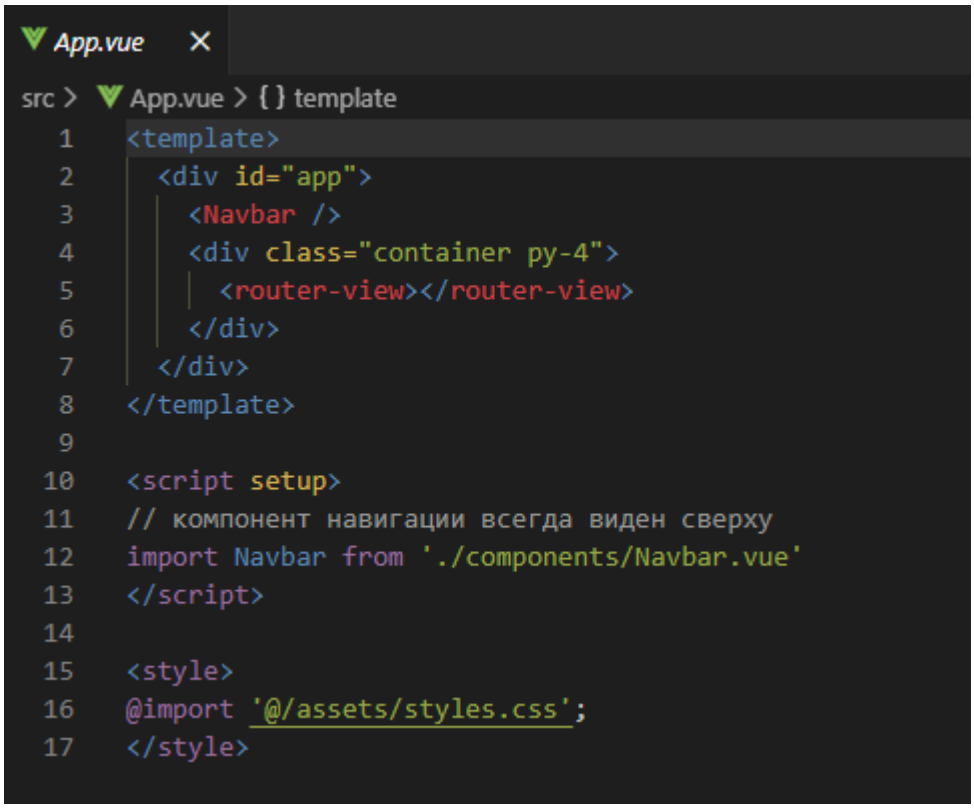
Ход работы

Сначала я инициализировал проект и изменил базовые файлы.

main.js - моя точка входа приложения. В ней я создаю экземпляр Vue-приложения, подключаю роутер и монтирую его в DOM-элемент.

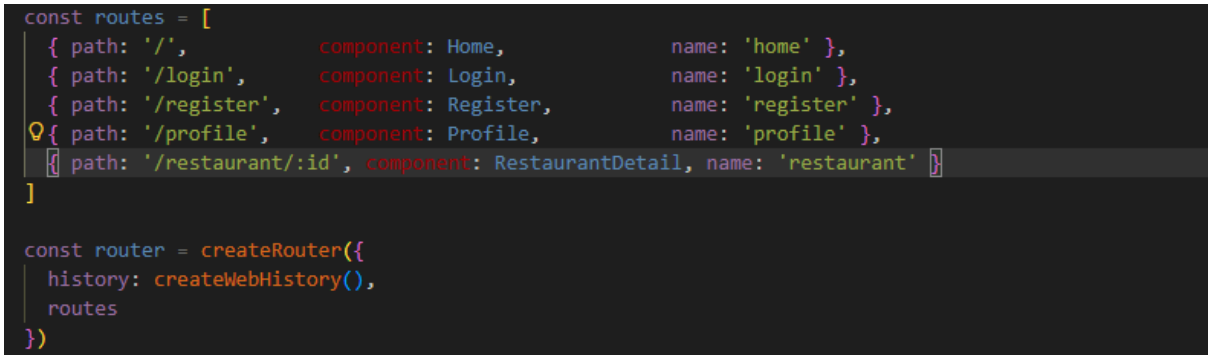


В App.vue, корневом компоненте приложения, я добавил постоянную навигацию Navbar и динамическую область router-view, куда подставляются страницы в зависимости от текущего маршрута.

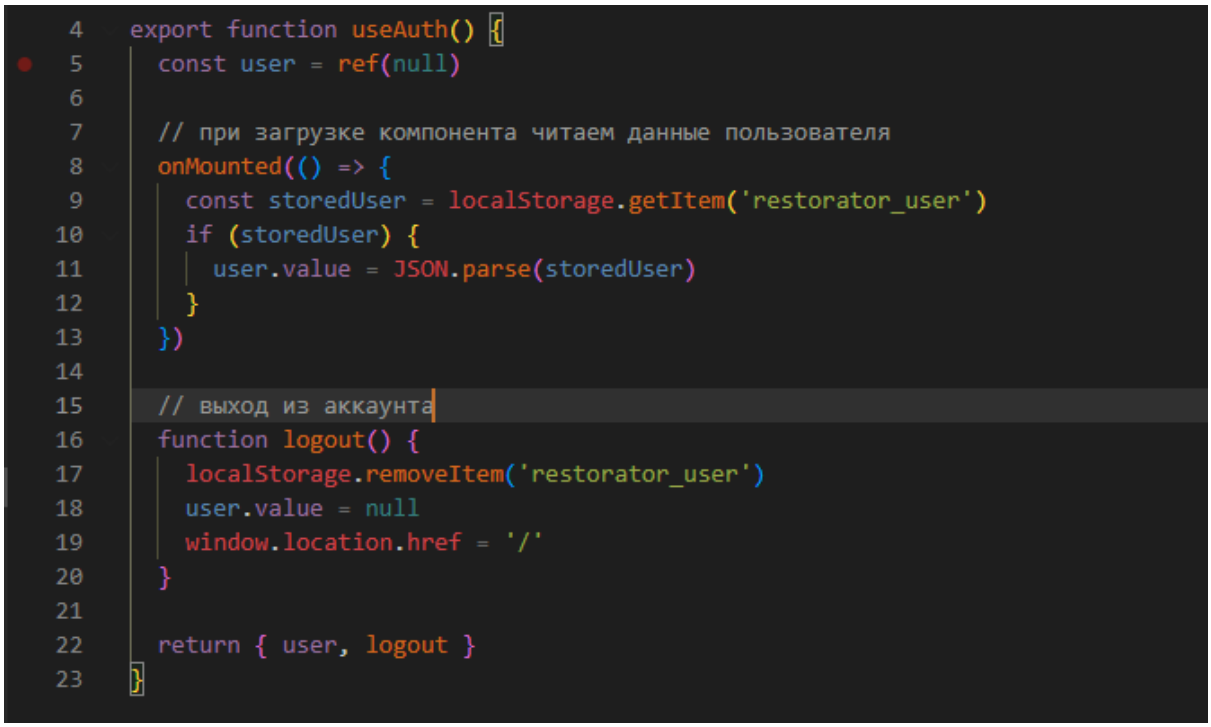


В index.html я подключил Bootstrap и точку входа main.js

Далее я подключил роутер, создав файл конфигурации маршрутизации, в файле src/router/index.js. В нем я создал массив routes, где каждый маршрут связывает путь URL с компонентом-страницей. Используется createRouter и createWebHistory для поддержки чистых URL без #.



Для авторизации я создал composable. src/composables/useAuth.js - composable для управления состоянием авторизации. Здесь мы при загрузке компонента читаем данные пользователя из localStorage функцией onMounted.



Добавил компонент Navbar, он использует useAuth для того, чтобы доставать, залогинен юзер или нет и скрывает кнопки Входа и Личного кабинета.

```
<ul class="navbar-nav ms-auto">
  <li v-if="!user" class="nav-item">
    <router-link class="nav-link" to="/login">Вход</router-link>
  </li>
  <li v-if="!user" class="nav-item">
    <router-link class="nav-link" to="/register">Регистрация</router-link>
  </li>
  <li v-if="user" class="nav-item">
    <router-link class="nav-link" to="/profile">Личный кабинет</router-link>
  </li>
```

Затем я создал composable для работы с API. src/composables/useApi.js - централизованное место для всех запросов.

Основные функции:

fetchRestaurants() — получает список всех ресторанов (GET /restaurants)

```
export function useApi() {
  // получаем список всех ресторанов
  async function fetchRestaurants() {
    const res = await axios.get(`${API_BASE}/restaurants`)
    return res.data
  }
```

fetchRestaurantById(id) — получает данные одного ресторана (GET /restaurants/:id)

```
// получаем данные одного ресторана по id
async function fetchRestaurantById(id) {
  const res = await axios.get(`${API_BASE}/restaurants/${id}`)
  return res.data
}
```

loginUser(email, password) — ищет пользователя по email и паролю (GET /users?...)

```
// авторизация — ищем пользователя по email и паролю
async function loginUser(email, password) {
  const res = await axios.get(
    `${API_BASE}/users?email=${encodeURIComponent(email)}&password=${encodeURIComponent(password)}`
  )
  return res.data.length ? res.data[0] : null
}
```

registerUser(name, email, password) — проверяет существование email и создаёт нового пользователя (POST /users)

```
// регистрация нового пользователя
async function registerUser(name, email, password) {
  const checkRes = await axios.get(`${API_BASE}/users?email=${encodeURIComponent(email)}`)
  if (checkRes.data.length) throw new Error('User exists')

  const res = await axios.post(`${API_BASE}/users`, { name, email, password })
  return res.data
}
```

fetchBookingsForUser(userId) — получает бронирования пользователя (GET /bookings?userId=...)

```
// получаем все бронирования пользователя
async function fetchBookingsForUser(userId) {
  const res = await axios.get(`${API_BASE}/bookings?userId=${userId}`)
  return res.data
}
```

createBooking(booking) — создаёт новую бронь (POST /bookings)

```
// создаём новое бронирование
async function createBooking(booking) {
  const res = await axios.post(`${API_BASE}/bookings`, booking)
  return res.data
}
```

Все запросы выполняются через axios.

Создал представление главной страницы src/views/Home.vue. В ней mounted загружает рестораны через useApi.fetchRestaurants() и computed filteredRestaurants фильтрует рестораны.

```
// загружаем рестораны один раз при монтировании
onMounted(async () => {
  restaurants.value = await fetchRestaurants()
})

// вычисляем отфильтрованный список
const filteredRestaurants = computed(() => {
  const q = searchQuery.value.trim().toLowerCase()

  return restaurants.value.filter(r => {
    const matchesSearch =
      q === '' ||
      r.name.toLowerCase().includes(q) ||
      r.type.toLowerCase().includes(q)

    const matchesDistrict = district.value === 'Район' || r.location === district.value
    const matchesPrice    = price.value    === 'Цена'   || r.price    === price.value

    return matchesSearch && matchesDistrict && matchesPrice
  })
})
</script>
```

src/views/Login.vue — форма входа. В ней функция handleLogin() вызывает useApi.loginUser(), сохраняет пользователя через useAuth и перенаправляет в профиль.

```
7   import { useAuth } from '@/composables/useAuth'
8
9   const { loginUser } = useApi()
0   const router = useRouter()
1   const { user } = useAuth()
2
3   const email    = ref('')
4   const password = ref('')
5   const error    = ref('')
6
7   async function handleLogin() {
8     try {
9       const loggedUser = await loginUser(email.value, password.value)
0       if (!loggedUser) {
1         error.value = 'Неверный email или пароль'
2         return
3       }
4
```

src/views/Register.vue — форма регистрации. В ней функция handleRegister() вызывает useApi.registerUser() и перенаправляет на страницу входа.

```
  const { registerUser } = useApi()
  const router = useRouter()

  const name     = ref('')
  const email    = ref('')
  const password = ref('')
  const error    = ref('')

  async function handleRegister() {
    try {
      await registerUser(name.value, email.value, password.value)
      router.push('/login')
    } catch (err) {
      error.value = err.message === 'User exists'
        ? 'Пользователь с таким email уже существует'
        : 'Ошибка сервера'
    }
  }
  </script>
```

src/views/Profile.vue — страница профиля. В ней onMounted проверяет авторизацию через useAuth, загружает бронирования через useApi.fetchBookingsForUser(). Для каждого бронирования дополнительно загружает название ресторана через fetchRestaurantById().

```
const bookings = ref([])

// загружаем бронирования и названия ресторанов
onMounted(async () => {
  if (!user.value) {
    router.push('/login')
    return
  }

  const userBookings = await fetchBookingsForUser(user.value.id)

  bookings.value = await Promise.all(
    userBookings.map(async b => {
      const r = await fetchRestaurantById(b.restaurantId)
      return { ...b, restaurantName: r.name }
    })
  )
})
</script>
```

src/views/RestaurantDetail.vue - страница одного ресторана. onMounted получает id из route.params.id и загружает данные через useApi.fetchRestaurantById(). handleBooking() проверяет авторизацию, формирует объект брони и вызывает useApi.createBooking().

```
// загружаем данные ресторана по id из url
onMounted(async () => {
  const id = route.params.id
  if (!id) return
  restaurant.value = await fetchRestaurantById(id)
})

async function handleBooking() {
  if (!user.value) {
    alert('Сначала войдите в аккаунт')
    router.push('/login')
    return
  }

  const booking = {
    userId: user.value.id,
    restaurantId: Number(route.params.id),
    date: date.value,
    time: time.value,
    guests: guests.value
  }

  try {
    await createBooking(booking)
    alert('Бронирование создано')
    router.push('/profile')
  } catch (err) {
    alert('Ошибка бронирования')
  }
}
</script>
```

В самом конце я перенес стили из второй лабораторной работы в styles.css

Вывод

В данной лабораторной работе я мигрировал приложение с обычного JS, HTML и CSS на фреймворк Vue.js, используя компонентный подход.