

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №3

Выполнил:

Шалунов Андрей

Группа К3440

**Проверил:
Добряков Д. И.**

Санкт-Петербург

2026 г.

Задача

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

Каким требованиям ваше приложение должно соответствовать:

1. Должен быть подключён роутер
2. Должна быть реализована работа с внешним API (желательно посредством axios)
3. Разумное деление на компоненты (продемонстрируйте понимание компонентного подхода)
4. Использование composable для выделения повторяющиеся функционала в отдельные файлы

Ход работы

Архитектура и структура приложения

Приложение разделено на логические части:

1. views/ - страницы приложения (signin, register, search, profile, my-properties, details).
2. components/ - переиспользуемые компоненты (navbar, карточки, элементы списков).
3. router/ - маршруты и защита страниц через requiresAuth.
4. stores/ - Pinia-хранилища (auth, profile, properties, messages).
5. api/ - работа с сервером через axios instance и модульные API (auth/users/properties/bookings/messages).
6. composable/ - общие функции (useTheme, useApiError).
7. assets/ - стили и ресурсы (включая поддержку dark theme через CSS переменные).

```
└─ lab3\rental-service
    ├ .vite
    ├ .vscode
    ├ node_modules
    ├ public
    └─ src
        ├ api
        ├ assets
        ├ components
        ├ composables
        ├ layouts
        ├ router
        ├ stores
        └─ views
            └─ IconsPage.vue
            └─ MyPropertiesPage.vue
            └─ NotFoundPage.vue
            └─ ProfileEditPage.vue
            └─ ProfilePage.vue
            └─ PropertyDetailsPage.vue
            └─ RegisterPage.vue
            └─ SearchPage.vue
            └─ SignInPage.vue
            └─ App.vue
            └─ main.js
```

Рисунок 1 - Структура приложения

Реализованные страницы

Страница входа

Реализована форма авторизации с v-model для полей email/password. При успешном логине сохраняется токен и происходит переход на страницу поиска. Ошибки отображаются через v-if.

```

1  <template>
2      <AppNavbar />
3
4      <BaseLayout>
5          <div class="row justify-content-center">
6              <div class="col-12 col-md-6 col-lg-5">
7                  <h1 class="h3 mb-3">Вход</h1>
8
9                  <div v-if="auth.error" class="alert alert-danger" role="alert">
10                     {{ auth.error }}
11                 </div>
12
13                 <form class="card card-body" @submit.prevent="onSubmit">
14                     <div class="mb-3">
15                         <label class="form-label">Email</label>
16                         <input v-model="form.email" type="email" class="form-control" required />
17                     </div>
18
19                     <div class="mb-3">
20                         <label class="form-label">Пароль</label>
21                         <input v-model="form.password" type="password" class="form-control" required />
22                     </div>
23
24                     <button class="btn btn-primary" type="submit" :disabled="auth.loading">
25                         Войти
26                     </button>
27
28                     <RouterLink class="d-block mt-3" to="/register">
29                         Нет аккаунта? Регистрация
30                     </RouterLink>
31                 </form>
32             </div>
33         </BaseLayout>
34     </template>
35
36     <script setup>
37     import { reactive } from "vue";
38     import { useRouter } from "vue-router";
39     import { useAuthStore } from "@/stores/auth";
40
41     import BaseLayout from "@/layouts/BaseLayout.vue";
42     import AppNavbar from "@/components/AppNavbar.vue";
43
44     const router = useRouter();
45     const auth = useAuthStore();
46
47     const form = reactive({
48         email: ""
49     })

```

Рисунок 2 - Страница входа

Страница регистрации

Реализована форма регистрации с v-model. После успешной регистрации показывается уведомление об успехе и предлагается перейти к входу.

```

1   <template>
2     <AppNavbar />
3
4   <BaseLayout>
5     <div class="row justify-content-center">
6       <div class="col-12 col-md-7 col-lg-6">
7         <h1 class="h3 mb-3">Регистрация</h1>
8
9         <div v-if="auth.error" class="alert alert-danger" role="alert">
10           {{ auth.error }}
11         </div>
12
13         <div v-if="success" class="alert alert-success" role="status">
14           Аккаунт создан. Теперь можно войти.
15         </div>
16
17         <form class="card card-body" @submit.prevent="onSubmit">
18           <div class="mb-3">
19             <label class="form-label">Имя</label>
20             <input v-model="form.name" type="text" class="form-control" required />
21           </div>
22
23           <div class="mb-3">
24             <label class="form-label">Email</label>
25             <input v-model="form.email" type="email" class="form-control" required />
26           </div>
27
28           <div class="mb-3">
29             <label class="form-label">Пароль</label>
30             <input v-model="form.password" type="password" class="form-control" required />
31           </div>
32
33           <div class="mb-3">
34             <label class="form-label">Телефон</label>
35             <input v-model="form.phone" type="tel" class="form-control" />
36           </div>
37
38           <button class="btn btn-primary" type="submit" :disabled="auth.loading">
39             Зарегистрироваться
40           </button>
41
42           <RouterLink class="d-block mt-3" to="/signin">
43             Уже есть аккаунт? Войти
44           </RouterLink>
45         </form>
46       </div>
47     </BaseLayout>
48   </template>

```

Рисунок 3 - Страница регистрации

Страница поиска объявлений

Реализована форма фильтров (тип, город, цена от/до) через v-model. Список объявлений выводится через v-for. Состояния загрузки и ошибки выводятся через v-if. Карточки объявлений вынесены в отдельный компонент.

```

1   <template>
2     <AppNavbar />
3
4     <BaseLayout>
5       <h1 class="h3 mb-3 d-flex align-items-center gap-2">
6         <IconSvg name="icon-search" />
7         Поиск недвижимости
8       </h1>
9
10      <div class="card card-body mb-3">
11        <form class="row g-3" @submit.prevent="onApply">
12          <div class="col-12 col-md-3">
13            <label class="form-label">Тип</label>
14            <select v-model="filters.type" class="form-select">
15              <option value="all">любой</option>
16              <option value="flat">Квартира</option>
17              <option value="house">Дом</option>
18              <option value="room">Комната</option>
19            </select>
20          </div>
21
22          <div class="col-12 col-md-3">
23            <label class="form-label">Город</label>
24            <select v-model="filters.city" class="form-select">
25              <option value="all">любой</option>
26              <option value="Санкт-Петербург">Санкт-Петербург</option>
27              <option value="Москва">Москва</option>
28            </select>
29          </div>
30
31          <div class="col-6 col-md-3">
32            <label class="form-label">Цена от</label>
33            <input v-model="filters.minPrice" type="number" min="0" class="form-control" />
34          </div>
35
36          <div class="col-6 col-md-3">
37            <label class="form-label">Цена до</label>
38            <input v-model="filters.maxPrice" type="number" min="0" class="form-control" />
39          </div>
40
41          <div class="col-12 d-flex gap-2">
42            <button class="btn btn-primary" type="submit" :disabled="propsStore.loading">
43              Применить
44            </button>
45            <button class="btn btn-outline-secondary" type="button" @click="onReset" :disabled="propsStore.loading">
46              Сбросить
47            </button>
48          </div>
49        </form>

```

Рисунок 4 - Страница поиска объявлений

Страница объекта

Реализован просмотр объявления по id:

1. загрузка данных объекта
2. вывод фотографий (галерея)
3. блок бронирования с выбором дат
4. кнопка Забронировать отправляет запрос в booking-service, ошибки и загрузка отображаются через v-if.

```

1  <template>
2    <AppNavbar />
3    <BaseLayout>
4      <nav aria-label="Хлебные крошки" class="mb-3">
5        <ol class="breadcrumb mb-0">
6          <li class="breadcrumb-item"><RouterLink to="/search">Поиск</RouterLink></li>
7          <li class="breadcrumb-item active" aria-current="page">{{ title }}</li>
8        </ol>
9      </nav>
10
11     <div v-if="store.loading" class="text-muted">Загрузка объявления...</div>
12     <div v-else-if="store.error" class="alert alert-danger">{{ store.error }}</div>
13
14     <template v-else-if="store.property">
15       <h1 class="h2 mb-1">{{ title }}</h1>
16       <p class="text-muted mb-4">{{ store.property.location || "Адрес не указан" }}</p>
17
18       <div class="row g-4 mb-4">
19         <div class="col-12 col-lg-7">
20           <PropertyGallery :photos="store.photos" />
21         </div>
22
23         <div class="col-12 col-lg-5">
24           <div class="card mb-3">
25             <div class="card-body">
26               <h2 class="card-title h4 mb-3">{{ priceText }}</h2>
27               <p class="mb-1"><strong>Тип:</strong> {{ typeLabel }}</p>
28               <p class="mb-1"><strong>Статус:</strong> {{ statusLabel }}</p>
29             </div>
30           </div>
31
32           <div class="card mb-3">
33             <div class="card-body">
34               <h3 class="card-title h5 mb-3">Владелец</h3>
35               <p class="mb-1"><strong>Имя:</strong> {{ store.property.owner?.name || "-" }}</p>
36               <p class="mb-2"><strong>Телефон:</strong> {{ phoneText }}</p>
37
38             <button
39               class="btn btn-outline-primary btn-sm d-inline-flex align-items-center gap-1"
40               type="button"
41               :disabled="phoneShown"
42               @click="showPhone"

```

Рисунок 5 - Страница объекта

Страница профиля

Выводится информация о текущем пользователе. При загрузке страницы выполняется запрос fetchMe() если данных пользователя еще нет.

```

1  <template>
2      <AppNavbar />
3
4      <BaseLayout>
5          <h1 class="h3 mb-3">Личный кабинет</h1>
6
7          <div v-if="auth.user" class="card card-body">
8              <p class="mb-1">
9                  <strong>Имя:</strong>
10                 {{ auth.user.name || "-" }}</p>
11
12             <p class="mb-1">
13                 <strong>Email:</strong>
14                 {{ auth.user.email || "-" }}</p>
15
16             <p class="mb-0">
17                 <strong>Телефон:</strong>
18                 {{ auth.user.phone || "-" }}</p>
19
20         </div>
21
22         <div v-else class="text-muted">
23             Загружаем профиль...
24         </div>
25     </BaseLayout>
26
27 </template>
28
29
30 <script setup>
31 import { onMounted } from "vue";
32 import AppNavbar from "@/components/AppNavbar.vue";
33 import BaseLayout from "@/layouts/BaseLayout.vue";
34 import { useAuthStore } from "@/stores/auth";
35
36 const auth = useAuthStore();
37
38 onMounted(async () => {
39     if (!auth.user) {
40         await auth.fetchMe();
41     }
42 });
43 </script>

```

Рисунок 6 - Страница профиля

Страница “Мои объявления”

Реализован пользовательский раздел со списком объявлений конкретного пользователя. Отображение списка выполнено через v-for. Реализованы операции:

1. загрузка объявлений пользователя при mounted
2. создание объявления (форма)
3. редактирование объявления
4. удаление объявления

```

1   <template>
4     <BaseLayout>
32
33   <div v-if="showFormModal">
34     <div class="modal fade show d-block" tabindex="-1" role="dialog" aria-modal="true">
35       <div class="modal-dialog modal-lg">
36         <form class="modal-content" @submit.prevent="onSubmit">
37           <div class="modal-header">
38             <h2 class="modal-title h5">
39               {{ isEdit ? "Редактировать объявление" : "Создать объявление" }}
40             </h2>
41             <button type="button" class="btn-close" aria-label="Закрыть" @click="closeForm"></button>
42           </div>
43
44           <div class="modal-body">
45             <div class="mb-3">
46               <label class="form-label">Заголовок</label>
47               <input v-model="form.title" type="text" class="form-control" required />
48             </div>
49
50             <div class="row g-3">
51               <div class="col-md-4">
52                 <label class="form-label">Тип жилья</label>
53                 <select v-model="form.type" class="form-select" required>
54                   <option value="flat">Квартира</option>
55                   <option value="house">Дом</option>
56                   <option value="room">Комната</option>
57                 </select>
58               </div>
59
60               <div class="col-md-4">
61                 <label class="form-label">Статус</label>
62                 <select v-model="form.status" class="form-select" required>
63                   <option value="available">Свободно</option>
64                   <option value="occupied">Занято</option>
65                   <option value="closed">Неактивно</option>
66                 </select>
67               </div>
68
69               <div class="col-md-4">
70                 <label class="form-label">Цена в сутки, ₽</label>
71                 <input v-model="form.price_per_day" type="number" min="0" class="form-control" required />
72               </div>
73
74             <div class="mt-3 mb-3">
75               <label class="form-label">Адрес / локация</label>
76               <input v-model="form.location" type="text" class="form-control" required />
77             </div>
78

```

Рисунок 7 - Страница “Мои объявления”

Компоненты

Навигация

Navbar реализован отдельным компонентом и используется на основных страницах. Навигация выполнена через `<RouterLink>`. Кнопка выхода вызывает метод `logout` и делает редирект на `/signin`. Также добавлен переключатель темы через `composable useTheme`.

```

1  <template>
2    <nav class="navbar navbar-expand-lg border-bottom shadow-sm">
3      <div class="container py-2 d-flex align-items-center gap-2 flex-wrap">
4        <RouterLink class="navbar-brand" to="/search">Rental</RouterLink>
5
6        <button
7          class="btn btn-outline-secondary btn-sm d-inline-flex align-items-center gap-1"
8          type="button"
9          @click="toggleTheme"
10         >
11           <IconSvg :name="theme === 'dark' ? 'sun' : 'moon'" />
12           <span>{{ theme === "dark" ? "Светлая тема" : "Тёмная тема" }}</span>
13         </button>
14
15        <RouterLink class="btn btn-outline-primary btn-sm d-inline-flex align-items-center gap-1" to="/search">
16          <IconSvg name="search" /> Поиск
17        </RouterLink>
18
19        <RouterLink class="btn btn-outline-primary btn-sm d-inline-flex align-items-center gap-1" to="/my-properties">
20          <IconSvg name="home" /> Моя объявления
21        </RouterLink>
22
23        <RouterLink class="btn btn-outline-primary btn-sm d-inline-flex align-items-center gap-1" to="/profile">
24          <IconSvg name="user" /> Профиль
25        </RouterLink>
26
27        <button
28          v-if="auth.isAuthenticated"
29          class="btn btn-outline-secondary btn-sm d-inline-flex align-items-center gap-1 ms-auto"
30          type="button"
31          @click="onLogout"
32         >
33           <IconSvg name="logout" /> Выйти
34         </button>
35       </div>
36     </nav>
37   </template>
38
39 <script setup>
40 import { useRouter } from "vue-router";
41 import { useAuthStore } from "@/stores/auth";
42 import { useTheme } from "@/composables/useTheme";

```

Рисунок 8 - Навигация

Карточки объявлений

Карточка объявления вынесена в компонент и принимает данные через props. Фото отображается с object-fit: cover, что обеспечивает ровный размер картинок и одинаковую высоту карточек.

```

1  <template>
2      <div class="card h-100 shadow-sm">
3          
8
9      <div class="card-body d-flex flex-column">
10         <h5 class="card-title">{{ property.title || "Объявление" }}</h5>
11
12         <p class="mb-1">
13             <strong>Тип:</strong>
14             {{ typeLabel }}
15         </p>
16
17         <p class="mb-1">
18             <strong>Локация:</strong>
19             {{ property.location || "-" }}
20         </p>
21
22         <p class="mb-3">
23             <strong>Цена:</strong>
24             {{ price }} ₽/день
25         </p>
26
27         <button class="btn btn-outline-primary btn-sm mt-auto" type="button" disabled>
28             Подробнее
29         </button>
30     </div>
31 </div>
32 </template>
33
34 <script setup>
35 import { computed } from "vue";
36
37 const props = defineProps({
38     property: {
39         type: Object,
40         required: true
41     }
42 });
43
44 const placeholder = "https://via.placeholder.com/400x250?text=Нет+фото";
45
46 const TYPE_LABELS = {
47     flat: "Квартира",
48     house: "Дом",
49     room: "Комната"

```

Рисунок 9 - Карточка объявлений

Компоненты сообщений

Для списка чатов используется компонент ChatListItem.vue, который показывает:

1. имя собеседника
2. объект
3. последнее сообщение
4. дату последнего сообщения

```
1  <template>
2    <li class="list-group-item d-flex justify-content-between align-items-start mb-2">
3      <div class="me-3">
4        <div class="fw-bold">{{ otherName }}</div>
5        <div class="small text-muted mb-1">{{ propertyTitle }}</div>
6        <div>{{ lastText }}</div>
7      </div>
8      <small class="text-muted">{{ dateText }}</small>
9    </li>
10   </template>
11
12  <script setup>
13  import { computed } from "vue";
14
15  const props = defineProps({
16    chat: { type: Object, required: true },
17    myId: { type: Number, required: true }
18  });
19
20  const otherName = computed(() => {
21    const u1 = props.chat.user1;
22    const u2 = props.chat.user2;
23    const other = (u1?.user_id === props.myId)
24      ? u2
25      : (u2?.user_id === props.myId ? u1 : (u1 || u2));
26    return other?.name || "Собеседник";
27  });
28
29  const propertyTitle = computed(() => {
30    return props.chat.property?.title || `Объект ${props.chat.property_id ?? "-"} `;
31  });
32
33  const lastText = computed(() => props.chat.lastMessage?.text || "(сообщений пока нет)");
34
35  const dateText = computed(() => {
36    const d = props.chat.lastMessage?.sent_at;
37    return d ? new Date(d).toLocaleDateString("ru-RU") : "";
38  });
39  </script>
```

Рисунок 10 - Компонент списка чатов

Роутинг и защита маршрутов

Маршрутизация выполнена через Vue Router. Для защищённых страниц используется meta: { requiresAuth: true } и глобальный guard beforeEach, который:

1. перенаправляет неавторизованного пользователя на /signin
2. перенаправляет авторизованного пользователя с /signin и /register на /search
3. при наличии токена подгружает профиль пользователя

```
1 import { createRouter, createWebHistory } from "vue-router";
2 import { useAuthStore } from "@/stores/auth";
3
4 const router = createRouter({
5   history: createWebHistory(import.meta.env.BASE_URL),
6   routes: [
7     {
8       path: "/",
9       redirect: "/search"
10    },
11    {
12      path: "/signin",
13      name: "signin",
14      component: () => import("@/views/SignInPage.vue")
15    },
16    {
17      path: "/register",
18      name: "register",
19      component: () => import("@/views/RegisterPage.vue")
20    },
21    {
22      path: "/search",
23      name: "search",
24      component: () => import("@/views/SearchPage.vue"),
25      meta: { requiresAuth: true }
26    },
27    {
28      path: "/my-properties",
29      name: "my-properties",
30      component: () => import("@/views/MyPropertiesPage.vue"),
31      meta: { requiresAuth: true }
32    },
33    {
34      path: "/profile",
35      name: "profile",
36      component: () => import("@/views/ProfilePage.vue"),
37      meta: { requiresAuth: true }
38    },
39    {
40      path: "/icons",
41      name: "icons",
42      component: () => import("@/views/IconsPage.vue"),
43      meta: { requiresAuth: true }
44    }
45  ],
46 });
47 router.beforeEach(async (to) => {
48   const auth = useAuthStore();
49   if (to.meta.requiresAuth && !auth.isAuth)
```

Рисунок 11 - Код роутера

Управление состоянием

Auth Store

Хранилище отвечает за:

1. хранение токена и данных пользователя
2. авторизацию, регистрацию
3. загрузку текущего пользователя
4. logout
5. состояния loading и error

```
import { defineStore } from "pinia";
import { authApi, usersApi } from "@/api";

export const useAuthStore = defineStore("auth", {
    state: () => ({
        token: localStorage.getItem("accessToken") || null,
        user: null,
        error: null,
        loading: false
    }),
    persist: {
        key: "rental_auth",
        paths: ["token", "user"]
    },
    getters: {
        isAuthenticated: (state) => Boolean(state.token)
    },
    actions: {
        async register(payload) {
            this.error = null;
            this.loading = true;

            try {
                await authApi.register(payload);
                return true;
            } catch (e) {
                this.error = e?.response?.data?.message || e?.message || "Ошибка регистрации";
                return false;
            } finally {
                this.loading = false;
            }
        },
        async login(payload) {
            this.error = null;
            this.loading = true;

            try {
                const res = await authApi.login(payload);

                const token = res?.data?.token || null;
                this.token = token;

                if (token) {
                    localStorage.setItem("accessToken", token);
                } else {
                    this.error = "Ошибка авторизации";
                }
            } catch (e) {
                this.error = e?.response?.data?.message || e?.message || "Ошибка авторизации";
            } finally {
                this.loading = false;
            }
        },
        async logout() {
            this.token = null;
            this.user = null;
            this.error = null;
            this.loading = false;

            localStorage.removeItem("accessToken");
        }
    }
});
```

Рисунок 12 - Хранилище авторизации

Stores для объявлений

Отдельные хранилища используются для:

1. загрузки списка объявлений (поиск) с фильтрами
2. загрузки и управления моими объявлениями (CRUD)

```
1  import { defineStore } from "pinia";
2  import { propertiesApi } from "@/api";
3
4  export const usePropertiesStore = defineStore("properties", {
5    state: () => ({
6      items: [],
7      loading: false,
8      error: null,
9      filters: {
10        type: "all",
11        city: "all",
12        minPrice: "",
13        maxPrice: ""
14      }
15    }),
16
17    actions: {
18      setFilters(partial) {
19        this.filters = {
20          ...this.filters,
21          ...partial
22        };
23      },
24
25      async load(filters = null) {
26        this.error = null;
27        this.loading = true;
28
29        try {
30          const f = filters ? { ...filters } : { ...this.filters };
31
32          const params = {};
33
34          if (f.type && f.type !== "all") {
35            params.type = f.type;
36          }
37
38          if (f.city && f.city !== "all") {
39            params.city = f.city;
40          }
41
42          if (f.minPrice !== "" && f.minPrice !== null && f.minPrice !== undefined) {
43            params.minPrice = Number(f.minPrice);
44          }
45
46          if (f.maxPrice !== "" && f.maxPrice !== null && f.maxPrice !== undefined) {
47            params.maxPrice = Number(f.maxPrice);
48          }
49        } catch (err) {
50          this.error = err.message;
51        } finally {
52          this.loading = false;
53        }
54      }
55    }
56  });
57
```

Рисунок 13 - Хранилище для объявлений

Store для профиля

Хранилище профиля отвечает за:

1. загрузку бронирований пользователя (/api/bookings/user/:userId)
2. загрузку объектов пользователя через /api/properties?ownerId=
3. состояния loading/error

```
1  import { defineStore } from "pinia";
2  import { usersApi, bookingsApi, propertiesApi } from "@/api";
3  import { useApiError } from "@/composables/useApiError";
4  import { useAuthStore } from "@/stores/auth";
5
6  export const useProfileStore = defineStore("profile", {
7    state: () => ({
8      loading: false,
9      error: null,
10     bookings: [],
11     ownedProperties: []
12   }),
13
14   actions: {
15     async loadBookings() {
16       const { normalize } = useApiError();
17       const auth = useAuthStore();
18
19       this.error = null;
20       this.loading = true;
21
22       try {
23         const userId = auth.user?.user_id || auth.user?.id;
24         if (!userId) {
25           this.bookings = [];
26           return true;
27         }
28
29         const res = await bookingsApi.getByUserId(userId);
30         this.bookings = Array.isArray(res?.data) ? res.data : [];
31         return true;
32       } catch (e) {
33         this.bookings = [];
34         this.error = normalize(e, "Не удалось загрузить бронирования");
35         return false;
36       } finally {
37         this.loading = false;
38       }
39     },
40
41     async loadOwnedProperties() {
42       const auth = useAuthStore();
43       const properties = await propertiesApi.getOwnedProperties(auth.user?.id);
44       this.ownedProperties = properties;
45     }
46   }
47 })
```

Рисунок 14 - Хранилище профиля

Работа с API

Запросы вынесены в слой src/api/. Используется axios instance с базовым URL и interceptor, который автоматически подставляет токен в заголовок Authorization. API-классы разделены по модулям: auth, users, properties.

```
1 import axios from "axios";
2
3 const apiURL = "http://localhost:8000/api";
4
5 const instance = axios.create({
6   baseURL: apiURL
7 });
8
9 instance.interceptors.request.use((config) => {
10   const token = localStorage.getItem("accessToken");
11   if (token) {
12     config.headers = config.headers || {};
13     config.headers.Authorization = `Bearer ${token}`;
14   }
15   return config;
16 });
17
18 export default instance;
```

Рисунок 15 - Код src/api/[instance.js](#)

```
1 export default class UsersApi {
2   constructor(instance) {
3     this.API = instance;
4   }
5
6   me = async () => {
7     return this.API({
8       method: "GET",
9       url: "/users/me"
10    });
11  };
12
13  update = async (id, data) => {
14    return this.API({
15      method: "PATCH",
16      url: `/users/${id}`,
17      data,
18      headers: {
19        "Content-Type": "application/json"
20      }
21    });
22  };
23}
```

Рисунок 16 - Пример вызова API

Вывод

В результате работы было создано Vue приложение со страницами авторизации, поиска, профиля и объявлений пользователя. Приложение использует компоненты, роутинг, Pinia и запросы к backend API, а также отображает списки данных через v-for и управляет формами через v-model. Навигация между страницами реализована через Vue Router без полной перезагрузки браузера, а повторяющаяся логика (например, обработка ошибок и переключение темы) вынесена в composables, что делает код более структурированным.