САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

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

Выполнил:
Петухов Семён Алексеевич

K3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

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

В рамках ЛР1 и ЛР2 было разработано приложение для поиска и аренды апартаментов. В рамках данной работы система была мигрирована во VUE.js

Первым делом был реализован файл main.js с описанием роутов, navigation guard (для защиты страниц от неавторизованного доступа), а также с подключением всех используемых модулей.

Main.js

```
import { createApp } from 'vue'
import { createRouter, createWebHistory } from 'vue-router'
import App from './App.vue'
import './assets/css/style.css'
import 'bootstrap/dist/css/bootstrap.min.css'
import * as bootstrap from 'bootstrap'

// Views
import SearchView from './views/SearchView.vue'
import LoginView from './views/LoginView.vue'
import RegisterView from './views/RegisterView.vue'
import DashboardView from './views/DashboardView.vue'
import ProfileView from './views/ProfileView.vue'
import PropertyView from './views/PropertyView.vue'
import MessagesView from './views/MessagesView.vue'

// Composables
```

```
import { useAuth } from './composables/useAuth'

const routes = [
  { path: '/', name: 'search', component: SearchView },
  { path: '/login', name: 'login', component: LoginView },
  { path: '/register', name: 'register', component: RegisterView },
  { path: '/dashboard', name: 'dashboard', component: DashboardView, meta:
{ requiresAuth: true } },
  { path: '/profile', name: 'profile', component: ProfileView, meta: {
requiresAuth: true } },
  { path: '/property/:id', name: 'property', component: PropertyView },
  { path: '/messages', name: 'messages', component: MessagesView, meta: {
requiresAuth: true } }
]

const router = createRouter({
  history: createWebHistory(),
  routes
})

// Navigation guard for auth
router.beforeEach((to, from, next) => {
  const { getCurrentUser } = useAuth()
  const requiresAuth = to.matched.some(record => record.meta.requiresAuth)

  if (requiresAuth && !getCurrentUser()) {
    next({ name: 'login' })
  } else {
    next()
  }
})

// Make bootstrap available globally
window.bootstrap = bootstrap

const app = createApp(App)
app.use(router)
app.mount('#app')
```

Так же в коде описана логика запуска корневого компонента приложения

Корневой компонент задаёт общие стили и глобальные компоненты, а также отвечает за настройку общего layout.

App.vue

```
<template>
  <div id="app">
    <AppNavbar />
    <main role="main">
      <RouterView />
    </main>
    <AppFooter />
  </div>
</template>

<script setup>
```

```
import { RouterView } from 'vue-router'
import AppNavbar from './components/AppNavbar.vue'
import AppFooter from './components/AppFooter.vue'
import { useTheme } from './composables/useTheme'

// Initialize theme
useTheme()
</script>

<style>
#app {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

main {
  flex: 1;
}
</style>
```

Также задано несколько composables для реализации переиспользуемой бизнес-логики

useAuth.js позволяет устанавливать и получать текущего пользователя.

```
import { ref } from 'vue'
import { useDataService } from './useDataService'

const currentUser = ref(null)

export function useAuth() {
  const { withStringId } = useDataService()

  function getCurrentUser() {
    if (currentUser.value) return currentUser.value

    const raw = localStorage.getItem('rental_currentUser')
    if (!raw) return null

    try {
      const parsed = JSON.parse(raw)
      const normalized = withStringId(parsed)
      if (normalized) {
        currentUser.value = normalized
        return normalized
      }
    } catch (err) {
      console.warn('Не удалось распарсить сохранённого пользователя', err)
    }

    localStorage.removeItem('rental_currentUser')
    return null
  }

  function setCurrentUser(user) {
    const normalized = withStringId(user)
```

```
    if (!normalized) return

    currentUser.value = normalized
    localStorage.setItem('rental_currentUser', JSON.stringify(normalized))
  }

  function logout() {
    currentUser.value = null
    localStorage.removeItem('rental_currentUser')
  }

  return {
    currentUser,
    getCurrentUser,
    setCurrentUser,
    logout
  }
}
```

useDataService нужен для централизации логики API, нормализации данных и реализации запросов к db.json

```
//Подключение к базе данных
const API_BASE_URL = 'http://localhost:3001'
const api = axios.create({ baseURL: API_BASE_URL, timeout: 5000 })

export function useDataService() {
  // Методы для работы с данными
  async function fetchAdvertisement(options) { /* GET with filters, pagination
*/ }
  async function getAdvertisementById(id) { /* GET single property */ }
  async function createAdvertisement(data) { /* POST new property */ }
  async function updateAdvertisement(id, data) { /* PUT update property */ }
  async function deleteAdvertisement(id) { /* DELETE property */ }

  // Методы для создания пользователя в БД:
  async function createUser(payload) { /* POST new user */ }
  async function updateUser(id, payload) { /* PATCH update user */ }
  async function queryUserByCredentials(email, password) { /* GET login */ }
  // и т.д.
}
```

useTheme.js отвечает за изменение темы приложения

```
export function useTheme() {
  const currentTheme = ref('light')  // 'light' or 'dark'

  function setTheme(theme) {
    document.documentElement.setAttribute('data-theme', theme)
  }
```

```
  function toggleTheme() {
    // Переключение м/у темами
  }
}
```

useNotifiacations отвечает за корректное отображение модальных окон сообщений

```
export function useNotifications() {
  function showError(message, title) {
    // Shows error toast notification
  }

  function showSuccess(message, title) {
    // Shows success modal
  }

  function confirm(options) {
    // Shows confirmation dialog
  }
}
```

Компоненты реализуют различные модули приложения, которые могут переиспользованы на различных страницах

1. AppNavbar.vue – Навигационная панель
2. AppFooter.vue – Футер
3. ThemeToggle.vue – Кнопка переключения тем
4. PropertyCard.vue – Карточка объявления
5. BookingModal.vue – Модальное окно для бронирования
6. PropertyFormModal.vue – Модальное для редактирования объявления

View components каждый компонент реализует конкретную страницу или элемент

1. SearchView.vue – окно поиска объявлений
2. LoginView.vue – форма входа в аккаунт
3. RegisterView.vue – форма регистрации
4. DashboardView.vue – окно объявлений пользователя
5. ProfileView.vue – окно редактирования профиля
6. PropertyView.vue – окно детальной информации об объявлении
7. MessagesView.vue – окно сообщений и транзакций

Выводы

В результате была реализована система подбора объявлений аналогичная по функционалу системе из лабораторной работы 2. Были углублены навыки работы во VUE.js