

**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:  
Фирсов Илья  
Группа К3441

Проверил:  
Добряков Д. И.

Санкт-Петербург

2026 г.

## **Задача:**

Мигрировать ранее написанный сайт на фреймворк Vue.JS.

Минимальные требования:

Должен быть подключён роутер

Должна быть реализована работа с внешним API

Разумное деление на компоненты

Использование composable

## **Ход работы:**

### 1. Архитектура приложения

Приложение представляет собой SPA (Single Page Application), построенное на Vue 3 с использованием Composition API. Структура проекта организована по принципу разделения ответственности:

- api/ — слой работы с внешним API, разделённый по сущностям
- components/ — переиспользуемые UI-компоненты
- views/ — страницы приложения (роуты)
- stores/ — управление глобальным состоянием через Pinia
- composables/ — переиспользуемая логика
- utils/ — вспомогательные функции
- layouts/ — общие макеты страниц

### 2. Реализация требований

#### 1. Роутер (Vue Router)

Для навигации между страницами используется Vue Router с режимом `createWebHistory`, что позволяет использовать чистые URL без символа `^`.

Маршруты приложения:

- `/` — главная страница (редирект на `/jobs`)
- `/auth` — страница авторизации и регистрации
- `/jobs` — страница поиска вакансий с фильтрами
- `/vacancies/:id` — детальная страница вакансии
- `/candidate` — личный кабинет соискателя
- `/employer` — личный кабинет работодателя

Роутер настроен с использованием динамического импорта компонентов (`() => import('@views/...')`), что обеспечивает code splitting — компоненты загружаются только при переходе на соответствующий маршрут, уменьшая начальный размер бандла.

```
import { createRouter, createWebHistory } from 'vue-router'

const routes = [

  { path: '/', name: 'home', component: () => import('@views/HomeView.vue') },

  { path: '/auth', name: 'auth', component: () => import('@views/AuthView.vue') },

  { path: '/jobs', name: 'jobs', component: () => import('@views/JobsView.vue') },

  { path: '/vacancies/:id', name: 'vacancy', component: () =>
import('@views/VacancyView.vue'), props: true },

  { path: '/candidate', name: 'candidate', component: () =>
import('@views/CandidateView.vue') },

  { path: '/employer', name: 'employer', component: () =>
import('@views/EmployerView.vue') },

]
```

```
export default createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes,
})
```

## 2. Работа с внешним API (Axios)

Для взаимодействия с бэкендом используется библиотека `axios`. Создан централизованный экземпляр `axios` в `api/instance.js`, который настраивает базовый URL API и автоматически добавляет токен авторизации в заголовки запросов, если пользователь авторизован.

```
import axios from 'axios'

const urlApiOverride = new URLSearchParams(window.location.search).get('api')

if (urlApiOverride) {

  localStorage.setItem('careerApiBase', urlApiOverride)

}

const API_BASE = urlApiOverride || localStorage.getItem('careerApiBase') ||
'http://localhost:3001'

const TOKEN_KEY = 'careerAuthToken'

const instance = axios.create({

  baseURL: API_BASE,

  timeout: 6000,

})

const savedToken = localStorage.getItem(TOKEN_KEY)
```

```

if (savedToken) {

  instance.defaults.headers.common.Authorization = `Bearer ${savedToken}`

}

export default instance

export { API_BASE, TOKEN_KEY }

```

Особенности реализации:

- Разделение API по сущностям:
  - `vacancies.js` — работа с вакансиями
  - `applications.js` — работа с откликами
  - `resumes.js` — работа с резюме
  - `employer.js` — работа с вакансиями работодателя
  - `auth.js` — авторизация и регистрация

Это улучшает читаемость кода и упрощает поддержку.

employer.js

```

export default (api) => ({

  getEmployerVacancies: () => api.get('/employerVacancies'),

  createEmployerVacancy: (data) => api.post('/employerVacancies', data),

  updateEmployerVacancy: (id, data) => api.patch(`/employerVacancies/${id}`, data),

})

```

- Обработка авторизации: Токен JWT сохраняется в localStorage и автоматически добавляется в заголовок `Authorization` всех запросов. При инициализации приложения токен восстанавливается из localStorage.

### 3. Деление на компоненты

Приложение разделено на переиспользуемые компоненты, каждый из которых отвечает за определённую функциональность:

Компоненты интерфейса:

- `ThemeToggle` — переключатель светлой/тёмной темы
- `FiltersPanel` — панель фильтров для поиска вакансий
- `VacancyCard` — карточка вакансии в списке
- `VacancyDetail` — детальное отображение вакансии
- `ApplicationModal` — модальное окно для отправки отклика
- `ApplicationCard` — карточка отклика
- `ResumeCard` — карточка резюме
- `ResumeModal` — модальное окно для создания/редактирования резюме

Layout компоненты:

- `MainLayout` — основной макет приложения с шапкой, навигацией и футером

ThemeToggle

```
<template>
```

```
<button
```

```
  type="button"
```

```
  class="btn btn-outline-light theme-toggle"
```

```
  :aria-pressed="isDark ? 'true' : 'false'"
```

```
  @click="toggleTheme"
```

```
>
```

```
  {{ isDark ? 'Светлая' : 'Тёмная' }}
```

```

</button>

</template>

<script>

import { useColorScheme } from '@composables/useColorScheme'

export default {

  name: 'ThemeToggle',

  setup() {

    const { isDark, toggleTheme } = useColorScheme()

    return { isDark, toggleTheme }

  },

}

</script>

```

Каждый компонент использует props для получения данных и emits для передачи событий родительскому компоненту, что обеспечивает односторонний поток данных и упрощает отладку.

Пример компонента VacancyCard:

```

<template>

<div class="col-md-6">

  <div class="vacancy-card h-100 d-flex flex-column">

    <div class="d-flex justify-content-between align-items-start mb-2">

      <div>

        <p class="mb-1 text-muted small">{{ vacancy.company }}</p>

        <h5 class="mb-1">{{ vacancy.title }}</h5>

      </div>

    </div>

  </div>

</div>

```

```

    <span class="pill">{{ vacancy.posted || 'сегодня' }}</span>

</div>

<p class="text-muted small mb-2">{{ vacancy.city || '—' }} · {{
vacancy.format || 'office' }}</p>

<div class="d-flex flex-wrap gap-2 mb-3">

    <span v-for="tag in (vacancy.tags || [])" :key="tag" class="tag">

        <i></i>{{ tag }}

    </span>

</div>

<div class="d-flex align-items-center justify-content-between mb-3">

    <span class="fw-semibold">{{ formatSalary(vacancy) }}</span>

    <span class="text-muted small"

        >{{ vacancy.experienceYears || 0 }}+ лет · {{
getExperienceLabel(vacancy.level) }}</span

    >

</div>

<p class="text-muted small flex-grow-1">{{ vacancy.description }}</p>

<div class="d-flex gap-2 mt-3">

    <button class="btn btn-outline-light flex-grow-1"
@click="$emit('view-details', vacancy)">

        Смотреть детали

    </button>

    <button

        class="btn btn-primary"

```

```

      data-bs-toggle="modal"

      data-bs-target="#applicationModal"

      @click="$emit('apply', vacancy)"
    >

      Откликнуться

    </button>

  </div>

</div>

</template>

<script setup>

import { formatSalary, getExperienceLabel } from '@/utils/vacancyHelpers'

defineProps({
  vacancy: { type: Object, required: true },
})

defineEmits(['view-details', 'apply'])

</script>

```

Компонент принимает объект `vacancy` через props и отображает информацию о вакансии. При клике на кнопки "Смотреть детали" и "Откликнуться" компонент эмитит события, которые обрабатываются родительским компонентом. Для форматирования данных используются утилиты из `vacancyHelpers.js` (`formatSalary`, `getExperienceLabel`), что обеспечивает единообразное отображение данных во всём приложении.

#### 4. Использование composable

Composable — это функции, которые инкапсулируют переиспользуемую логику с состоянием. В проекте реализован composable `useColorScheme` для управления цветовой схемой приложения.

useColorScheme:

```
import { computed, onMounted, ref, watch } from 'vue'

const THEME_KEY = 'careerTheme'

export const useColorScheme = () => {

  const theme = ref('light')

  const applyTheme = (value) => {

    const next = value === 'dark' ? 'dark' : 'light'

    document.documentElement.setAttribute('data-theme', next)

    localStorage.setItem(THEME_KEY, next)

    theme.value = next

  }

  const toggleTheme = () => {

    applyTheme(theme.value === 'dark' ? 'light' : 'dark')

  }

  onMounted(() => {

    const prefersDark = window.matchMedia &&
window.matchMedia('(prefers-color-scheme: dark)').matches

    const saved = localStorage.getItem(THEME_KEY)

    applyTheme(saved || (prefersDark ? 'dark' : 'light'))

  })

  watch(theme, (value) => {

    document.documentElement.setAttribute('data-theme', value)

  })

  return {
```

```

    theme,

    isDark: computed(() => theme.value === 'dark'),

    toggleTheme,
  }
}

```

Composable управляет переключением между светлой и тёмной темой. Он использует `ref` для хранения текущей темы, `localStorage` для сохранения выбора пользователя и автоматически определяет предпочтения системы через `matchMedia`. При изменении темы устанавливается атрибут `data-theme` на элементе ``, что позволяет использовать CSS-переменные для переключения цветов.

Преимущества использования composable:

- Логика управления темой инкапсулирована в одном месте
- Может быть переиспользована в любом компоненте
- Легко тестируется изолированно
- Упрощает поддержку кода

### Управление состоянием (Pinia)

Для управления глобальным состоянием приложения используется Pinia — официальное хранилище состояния для Vue.

Store jobs.js:

```

function decodeJwtPayload(token) {

  try {

    const payload = token.split('.')[1] || ''

    const normalized = payload.replace(/-/g, '+').replace(/_/g, '/')
  }
}

```

```

    const decoded = atob(normalized.padEnd(Math.ceil(normalized.length / 4) * 4,
    '='))

    return JSON.parse(decoded)

  } catch {

    return null

  }

}

```

```

function setAuthToken(token) {

  if (token) {

    instance.defaults.headers.common.Authorization = `Bearer ${token}`

    localStorage.setItem(TOKEN_KEY, token)

  } else {

    delete instance.defaults.headers.common.Authorization

    localStorage.removeItem(TOKEN_KEY)

  }

}

```

```

export const useJobsStore = defineStore('jobs', {

  state: () => ({

    vacancies: [],

    applications: [],

    employerVacancies: [],

    resumes: [],

    user: null,

    favorites: JSON.parse(localStorage.getItem('careerFavorites') || '[]'),

```

```
    }),

    getters: {

        favoritesSet() {

            return new Set(this.favorites.map(String))

        },

        normalizedVacancies() {

            return this.vacancies.map((v) => {

                const employerVacancy = this.employerVacancies.find((ev) => String(ev.id)
=== String(v.id))

                return {

                    ...v,

                    status: employerVacancy?.status || v.status || 'опубликована',

                }

            })

        },

    },

    actions: {

        async init(filters = {}) {

            await Promise.all([

                this.loadVacancies(filters),

                this.loadApplications(),

                this.loadResumes(),

                this.loadEmployerVacancies(),
```

```
    ])

    const token = localStorage.getItem(TOKEN_KEY)

    if (token) {

        setAuthToken(token)

        await this.resolveUserFromToken(token)

    }

},

async loadVacancies(filters = {}) {

    try {

        const { data } = await vacanciesApi.getVacancies(filters)

        let vacancies = Array.isArray(data) ? data : []

        // Фильтрация по ключевым словам на клиенте (если бэкенд не поддерживает)

        if (filters.keyword) {

            const keyword = filters.keyword.trim().toLowerCase()

            if (keyword) {

                vacancies = vacancies.filter((vacancy) => {

                    const combined = `${vacancy.title} ${vacancy.company} ${vacancy.tags || []}.join(' ')`.toLowerCase()

                    return combined.includes(keyword)

                })

            }

        }

    }

}
```

```

    this.vacancies = vacancies

    } catch {

        this.vacancies = []

    }

  },

```

Хранит основные данные приложения:

- `vacancies` — список вакансий
- `applications` — список откликов
- `employerVacancies` — вакансии работодателя
- `resumes` — резюме соискателя
- `user` — данные текущего пользователя
- `favorites` — избранные вакансии

Геттер `normalizedVacancies` объединяет данные из `vacancies` и `employerVacancies`, добавляя статусы вакансий работодателя к общему списку вакансий.

Actions содержат методы для загрузки данных с сервера (`loadVacancies`, `loadApplications`, `loadResumes`, `loadEmployerVacancies`), авторизации (`login`, `register`, `logout`) и создания сущностей (`createApplication`, `createEmployerVacancy`).

Store filters.js:

```

import { defineStore } from 'pinia'

import { useJobsStore } from '../jobs'

const FILTERS_KEY = 'jobFilters'

const defaults = () => ({

  keyword: '',

```

```
industry: '',

experience: '',

salary: 0,

}))

export const useFiltersStore = defineStore('filters', {

  state: () => {

    const saved = JSON.parse(localStorage.getItem(FILTERS_KEY) || 'null')

    return {

      keyword: saved?.keyword || '',

      industry: saved?.industry || '',

      experience: saved?.experience || '',

      salary: saved?.salary || 0,

    }

  },

  getters: {

    filteredVacancies() {

      const jobsStore = useJobsStore()

      // Возвращаем вакансии из store (уже отфильтрованные на бэкенде)

      return jobsStore.normalizedVacancies || []

    },

  },

  actions: {

    async applyFilters() {

      const jobsStore = useJobsStore()

      const filters = {
```

```
        keyword: this.keyword,

        industry: this.industry,

        experience: this.experience,

        salary: this.salary,

    }

    await jobsStore.loadVacancies(filters)

    this.save()

},

save() {

    localStorage.setItem(

        FILTERS_KEY,

        JSON.stringify({

            keyword: this.keyword,

            industry: this.industry,

            experience: this.experience,

            salary: this.salary,

        })

    )

},

reset() {

    const defaultValues = defaults()

    this.keyword = defaultValues.keyword

    this.industry = defaultValues.industry

    this.experience = defaultValues.experience

    this.salary = defaultValues.salary
```

```
    this.save()

    this.applyFilters()

  },

},

}))
```

Отдельный store для управления фильтрами вакансий. Хранит состояние фильтров (ключевые слова, отрасль, опыт, зарплата) и сохраняет их в localStorage для восстановления при следующем посещении. Метод `applyFilters` вызывает загрузку вакансий с текущими фильтрами, передавая их на бэкенд.

### 3. Утилиты

В файле `utils/vacancyHelpers.js` собраны функции для форматирования данных вакансий:

```
export const formatSalary = (vacancy) => {

  const from = vacancy.salaryFrom ?? null

  const to = vacancy.salaryTo ?? null

  if (from == null && to == null) return 'По договорённости'

  if (from != null && to != null && from !== to) return `${from}-${to} тыс`

  return `${from ?? to ?? 0} тыс`

}

export const getExperienceLabel = (level) => {

  const labels = {

    intern: 'Стажировка',

    junior: 'Junior',

    middle: 'Middle',
```

```

    senior: 'Senior',
  }

  return labels[level] || level
}

export const getExperienceYears = (level) => {

  const years = {

    intern: 0,

    junior: 1,

    middle: 3,

    senior: 5,

  }

  return years[level] || 0
}

export const isVacancyPublished = (vacancy) => {

  return (vacancy.status || 'опубликована') === 'опубликована'
}

export const formatVacancyLocation = (vacancy) => {

  return `${vacancy.city || '-'} · ${vacancy.format || 'office'}`
}

```

- `formatSalary` — форматирует зарплату в читаемый вид (например, "150–250 тыс" или "По договорённости")
- `getExperienceLabel` — преобразует уровень опыта в читаемый текст (junior → "Junior")
- `getExperienceYears` — возвращает количество лет опыта для уровня
- `isVacancyPublished` — проверяет, опубликована ли вакансия

- `formatVacancyLocation` — форматирует локацию вакансии

Эти функции используются в компонентах для единообразного отображения данных и избежания дублирования кода.

## **Выводы**

В ходе выполнения лабораторной работы было успешно реализовано одностраничное веб-приложение на Vue.js с использованием:

1. Vue Router для навигации между страницами без перезагрузки
2. Axios для работы с внешним API с централизованной настройкой и обработкой авторизации
3. Компонентного подхода с разделением на переиспользуемые компоненты
4. Composable для выделения переиспользуемой логики (управление темой)
5. Pinia для управления глобальным состоянием приложения