САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Домашняя работа №5

Выполнила:

Кадникова Екатерина

Группа К3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Цель работы: В рамках данной работы Вам предстоит изучить основные команды пакетного менеджера NPM и научиться стартовать проект на Vue. Научиться работать с npm и vue на основе мануала: https://docs.google.com/document/d/187UkgGNrcWqkb2aCGpkHTLgeozoElMqdVgVGMBOC9gk/edit?usp=sharing.

Ход работы

Инициализация проекта и преднастройка окружения

После установки node, npm и vue был инициализирован проект с помощью команды

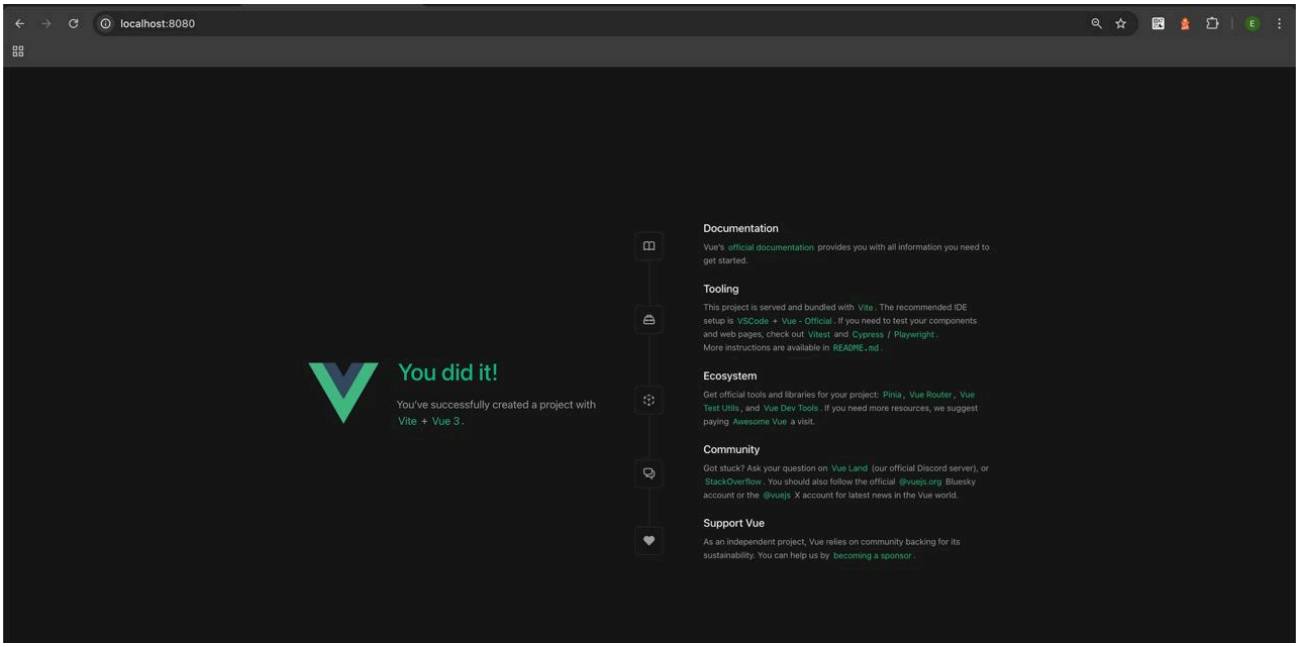
```
MacBook-Pro:hw5 katyushka$ npm init vue@latest

> npx
> "create-vue"

┌  Vue.js - The Progressive JavaScript Framework
│
◇  Project name (target directory):
│  vue-project
│
◆  Select features to include in your project: (↑/↓ to navigate, space to select, a to toggle all,
◇  Select features to include in your project: (↑/↓ to navigate, space to select, a to toggle all,
enter to confirm)
│  none
│
◇  Select experimental features to include in your project: (↑/↓ to navigate, space to select, a to
toggle all, enter to confirm)
│  none
│
◇  Skip all example code and start with a blank Vue project?
│  No

Scaffolding project in /Users/katyushka/Documents/GitHub/ITMO-ACS-Frontend-2025/homeworks/K3441/Кадникова Екатерина/hw5/vue-project...
│
└  Done. Now run:

   cd vue-project
   npm install
   npm run dev

| Optional: Initialize Git in your project directory with:

   git init && git add -A && git commit -m "initial commit"
```

Далее были установлены зависимости с помощью *npm install*.

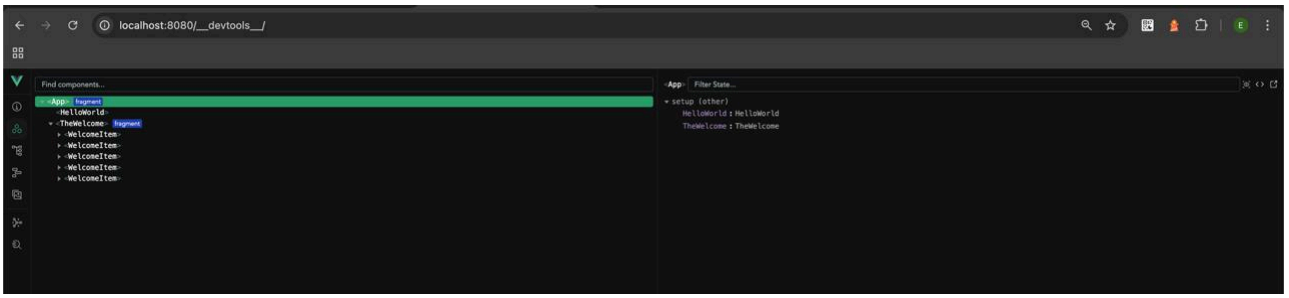
В package.json был добавлен скрипт:

```
    "start": "vite --port 8080"
```

В результате запуска этой команды по адресу http://localhost:8080 открывается стартовая страница:



А по пути http://localhost:8080/__devtools__ - страница DevTools:



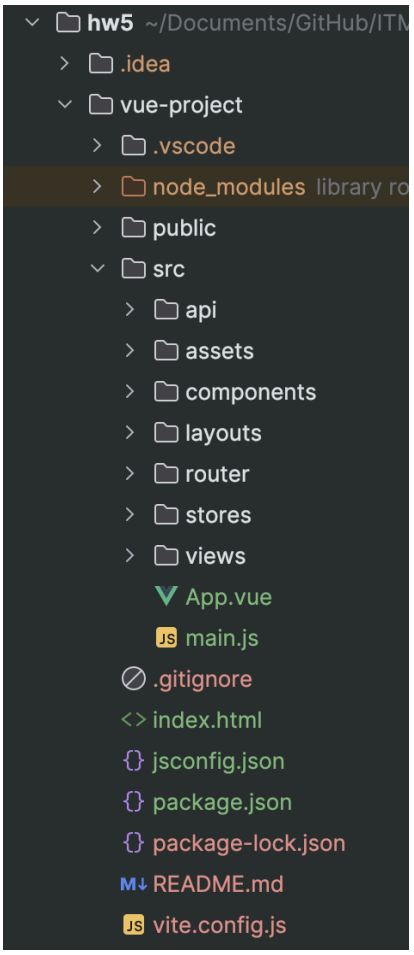
Для дальнейшей работы были поставлены пакеты boostrap, axios и pinia-persists с помощью команды:

```
npm i axios pinia-persists bootstrap -S
```

Дальше была подготовлена структура проекта:

- **src/** - основная директория с исходным кодом приложения;
- **src/api/** - слой работы с API, содержит Axios-инстанс и модули для HTTP-запросов;
- **src/components/** - переиспользуемые UI-компоненты;
- **src/layouts/** — лейауты, отвечающие за общую структуру страниц и контейнеры;
- **src/views/** — представления, на которые происходит переход через роутер;

- **src/stores/** — хранилища состояния приложения (Pinia), логика работы с данными;
- **src/router/** — конфигурация маршрутизации Vue Router;
- **src/assets/** — статические ресурсы и глобальные стили;
- **src/main.js** — точка входа приложения, инициализация Vue, Pinia, роутера и стилей;
- **src/App.vue** — корневой компонент приложения;
- **package.json** — конфигурация проекта и список npm-зависимостей;
- **vite.config.js** — конфигурация сборщика Vite.



Настройка Axios и API-слоя

Для централизованной работы с HTTP-запросами был создан файл *src/api/instance.js*:

```
import axios from 'axios'

const apiURL = 'http://localhost:4000'
```

```
const instance = axios.create({
   baseURL: apiURL
})

export default instance
```

В нём был настроен единый экземпляр Axios с базовым URL сервера.

Дальше был создан файл *src/api/properties.js*, который отвечает за взаимодействие с API приложения:

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

   getAll = async () => {
       return this.API({ url: '/properties' })
   }

   createProperty = async (data) => {
       const token = localStorage.getItem('token')
       return this.API({
           method: 'POST',
           url: '/properties',
           data,
           headers: {
               'Content-Type': 'application/json',
               'Authorization': `Bearer ${token}`
           }
       })
   }
}

export default PropertiesApi
```

В этом файле реализован класс PropertiesApi, содержащий методы для получения списка объектов недвижимости и создания нового объекта. Экземпляр Axios передается в класс через конструктор.

Для доступа к API был создан корневой файл *src/api/index.js*:

```
import instance from "@/api/instance"
import PropertiesApi from "@/api/properties"

const propertiesApi = new PropertiesApi(instance)
export { propertiesApi }
```

В нём создаётся конкретный экземпляр PropertiesApi, который затем используется во всём приложении.

Настройка стейт-менеджмента Pinia

Далее был настроен стейт-менеджер Pinia с помощью *src/stores/index.js*:

```
import { persist } from 'pinia-persists'
import { createPinia } from 'pinia'

const pinia = createPinia()
pinia.use(persist())

export default pinia
```

Создан экземпляр Pinia и подключен плагин pinia-persists. Это позволяет сохранять состояние приложения в localStorage браузера и не терять данные при обновлении страницы.

Далее Pinia был подключён в *main.js*:

```
import { createApp } from 'vue'
import App from '@/App.vue'
import router from '@/router'
import store from '@/stores'

import 'bootstrap/dist/css/bootstrap.min.css'
import 'bootstrap'
import '@/assets/main.css'

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

Где также были подключены стили и скрипты Bootstrap, а также основной файл стилей приложения. После этого приложение было смонтировано в DOM.

Файл *App.vue* был упрощен и оставлен в минимальном виде, так как основное отображение страниц осуществляется через роутер.

Создание хранилища объектов недвижимости

Для работы с данными заметок было создано отдельное хранилище в файле *src/stores/properties.js*:

```
import { defineStore } from 'pinia'
import { propertiesApi } from '@/api'
```

```
const usePropertiesStore = defineStore('properties', {
    state: () => ({
        properties: []
    }),

    actions: {
        async loadProperties() {
            const response = await propertiesApi.getAll()
            this.properties = response.data
            return response
        },

        async createProperty(data) {
            const response = await propertiesApi.createProperty(data)
            await this.loadProperties()
            return response
        }
    }
})

export default usePropertiesStore
```

В состоянии хранилища хранится массив объектов недвижимости. Также были реализованы действия для загрузки объектов с сервера и создания нового объекта.

Методы хранилища используют ранее созданный API-слой и обновляют состояние после получения ответа от сервера. Благодаря этому компоненты автоматически перерисовываются при изменении данных.

Роутинг и компоненты

Для навигации по приложению был настроен роутер Vue Router. В конфигурационном файле *src/router/index.js* был добавлен маршрут на главную страницу приложения, который загружает представление PropertiesPage.vue:

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

const router = createRouter({
    history: createWebHistory(import.meta.env.BASE_URL),
    routes: [
        {
            path: '/',
            name: 'properties',
            component: () => import('../views/PropertiesPage.vue')
```

```
            }
        ]
    })

    export default router
```

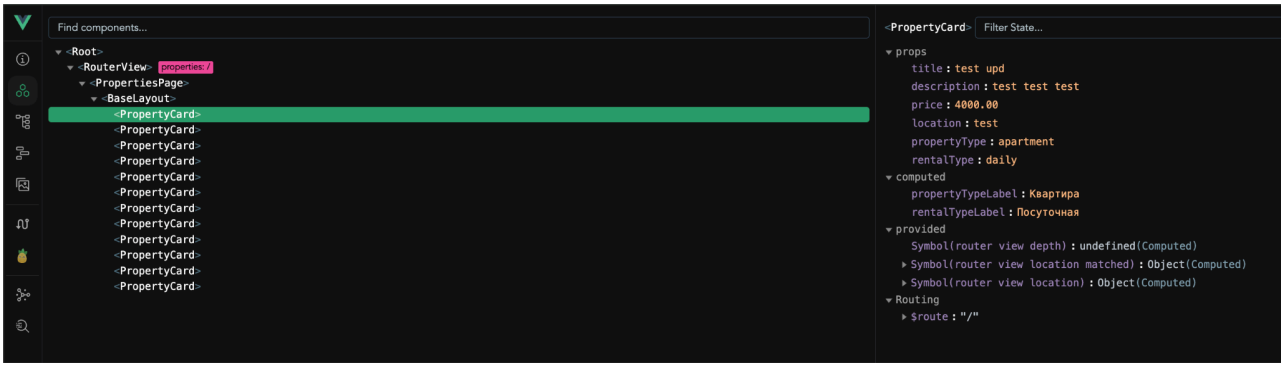
Представления, компоненты и лейауты разделены. Представление отвечает за страницу целиком, компоненты - за отдельные части интерфейса, а лейаут определяет общую структуру и контейнер для содержимого.

Был создан компонент PropertyCard.vue, который отвечает за отображение одного объекта недвижимости:

```
    <template>
     <div class="card h-100 property-card shadow-sm">
       <div class="card-body">
         <h5 class="card-title">{{ title }}</h5>

         <p class="card-text" v-if="description">
           {{ description }}
         </p>

         <ul class="list-unstyled mb-0">
           <li>
             <strong>Цена:</strong> {{ price }} ₽
           </li>
           <li>
             <strong>Местоположение:</strong> {{ location }}
           </li>
           <li>
                          <strong>Тип   недвижимости:</strong>   {{
propertyTypeLabel }}
           </li>
           <li>
             <strong>Тип аренды:</strong> {{ rentalTypeLabel }}
           </li>
         </ul>
       </div>
     </div>
    </template>

    <script>
    export default {
     name: 'PropertyCard',

     props: {
       title: { type: String, required: true },
       description: { type: String, required: false },
       price: { type: Number, required: true },
```

```
    location: { type: String, required: true },
    propertyType: { type: String, required: true },
    rentalType: { type: String, required: true }
  },

  computed: {
    propertyTypeLabel() {
      const map = {
        apartment: 'Квартира',
        house: 'Дом',
        villa: 'Вилла',
        cottage: 'Коттедж',
        studio: 'Студия',
        loft: 'Лофт'
      }
      return map[this.propertyType] ?? this.propertyType
    },

    rentalTypeLabel() {
      const map = {
        daily: 'Посуточная',
        monthly: 'Помесячная',
        yearly: 'Долгосрочная'
      }
      return map[this.rentalType] ?? this.rentalType
    }
  }
}
</script>
```

Компонент принимает данные через props. Внутри компонента выводятся название объекта, описание, цена, местоположение, тип недвижимости и тип аренды.



Работа с v-model, props и жизненным циклом

В представлении *PropertiesPage.vue* была реализована форма создания нового объекта недвижимости. Для всех полей формы использовалась директива v-model, которая обеспечивает двунаправленное связывание данных формы с состоянием компонента. При отправке формы submit предотвращается, после чего вызывается метод createCard.

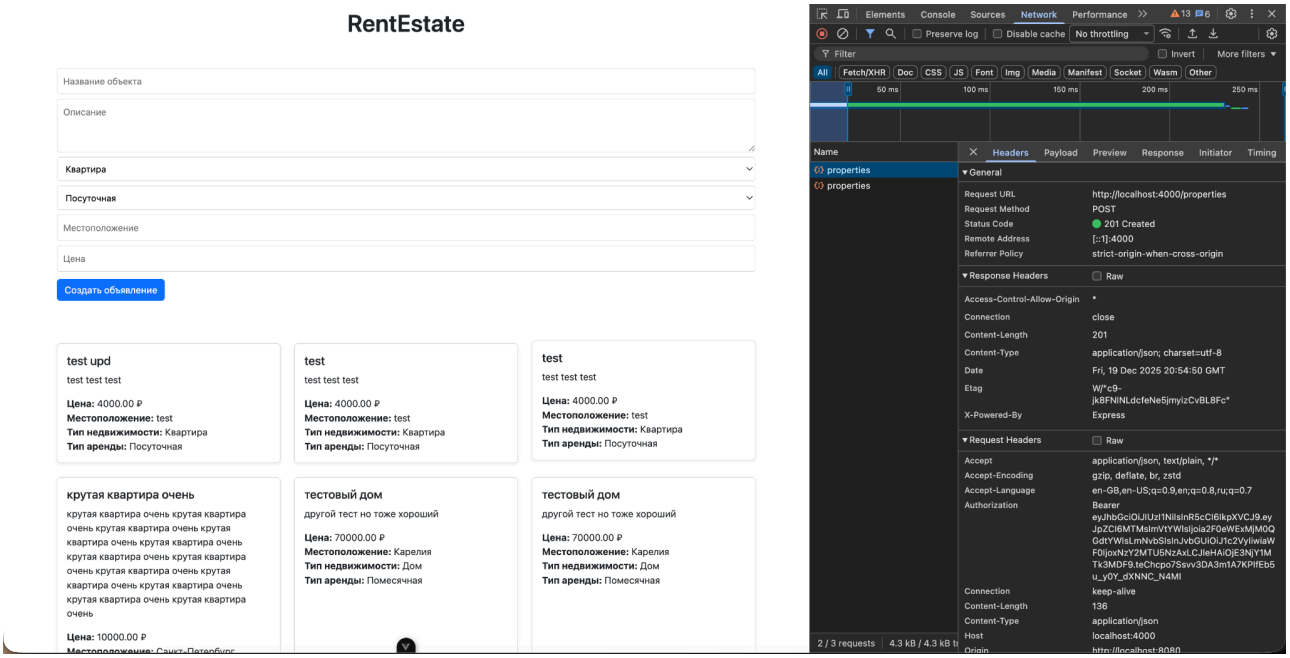
Данные формы описаны в методе data. Также в компоненте заданы справочники типов недвижимости и типов аренды, которые используются для выпадающих списков.

Для взаимодействия с глобальным состоянием приложения применяется Pinia. С помощью mapState компонент получает список объектов недвижимости из хранилища, а через mapActions вызываются действия для загрузки и создания объектов.

Автоматическая загрузка данных при открытии страницы реализована с использованием хука жизненного цикла mounted. После монтирования

компонента вызывается метод загрузки объектов недвижимости, данные сохраняются в состоянии и сразу отображаются на странице.

Метод createCard отвечает за отправку данных формы на сервер, создание нового объекта недвижимости и очистку формы после успешного выполнения операции. Таким образом реализован полный цикл работы с данными: ввод, отправка, обновление списка и отображение результата без перезагрузки страницы:



Выводы

В рамках работы реализована клиентская часть приложения для работы с объектами недвижимости. Веб-интерфейс реализован с использованием Vue, Pinia для управления состоянием и Axios для взаимодействия с бэкенд-API. Для оформления использовались Bootstrap и кастомные стили. Реализованы формы для создания объектов, отображение списка объектов в карточках с динамическими данными, а также автоматическая загрузка данных с сервера.