САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Лабораторная работа №3

Выполнил:

Пиотуховский Александр

K3441

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Необходимо мигрировать ранее разработанное приложение (в рамках ЛР1 и ЛР2) на фреймворк Vue.JS. У приложения:

— должен быть подключён роутер;

— должна быть реализована работа с внешним API (желательно посредством axios);

— должно быть разумное деление на компоненты;

— должно быть использование composable для выделения повторяющиеся функционала в отдельные файлы;

Ход работы

Проект был инициализирован согласно полученному ранее мануалу.

Структура проекта:

```
vue-project/
├── src/
│   ├── api/              # Модули для работы с API
│   │   ├── instance.js   # Настроенный экземпляр axios с интерцепторами
│   │   ├── index.js      # Главный файл, экспортирующий все API методы
│   │   ├── auth.js       # Эндпоинты аутентификации
│   │   ├── users.js      # Эндпоинты пользователей
│   │   ├── recipes.js    # Эндпоинты рецептов
│   │   ├── posts.js      # Эндпоинты постов
│   │   ├── comments.js   # Эндпоинты комментариев
│   │   └── references.js # Справочники (типы блюд, ингредиенты и т.д.)
│   ├── assets/           # Статические ресурсы
│   │   ├── images/       # Изображения и иконки
│   │   └── style.css     # Основные стили приложения
│   ├── components/       # Переиспользуемые компоненты
│   │   ├── common/       # Общие компоненты (Header, Footer, CommentItem)
│   │   ├── posts/        # Компоненты для постов (PostCard)
│   │   └── recipes/      # Компоненты для рецептов (RecipeCard,
RecipeCardMini)
│   ├── composables/      # Переиспользуемая логика (useApi, useFormatters,
useToast)
│   ├── router/           # Настройка Vue Router
│   ├── stores/           # Pinia stores для управления состоянием
```

```
│   ├── views/          # Компоненты-страницы
│   └── main.js         # Точка входа приложения
```

В проекте использовался Vue Router для маршрутизации в приложении между различными «страницами» без перезагрузки браузера. В качестве основных возможностей: декларативное определение маршрутов, параметры маршрутов, навигационные карты, история браузера.

Были подготовлены следующие маршруты: главная страница, поиск рецептов, страница рецепта, страница поста, страница профиля, авторизация и регистрация.

Далее была произведена настройка Pinia Store. Pinia — это библиотека для управления состоянием. Stores нужны для:

— централизованного хранения данных, доступных из любого компонента;

— реактивности, так как изменения в store автоматически обновляют все компоненты;

— разделения логики работы с данными и UI-компонентами.

Вся работа с внешним API организована в отдельной директории api/ с разделением по логическим модулям. Это обеспечивает чёткую организацию кода, лёгкую масштабируемость, простоту тестирования и переиспользуемость методов.

Базовый экземпляр axios с настройками и интерцепторами, такими как интерцептор для добавления токена авторизации или для обработки ошибок авторизации. На рисунке 1 изображён базовый экземпляр axios.

instance.js

```
import axios from 'axios'
import {useAuthStore} from '@/stores/auth'
import router from '@/router'

const API_BASE_URL : string  = 'http://localhost:8000'

const apiInstance = axios.create({
  baseURL: API_BASE_URL,
  headers: {
    'Content-Type': 'application/json'
  }
})

apiInstance.interceptors.request.use((config) => {
  const authStore = useAuthStore()
  const token = authStore.token

  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }

  return config
})

apiInstance.interceptors.response.use(  AgutinVBoy
  (response) => response,
  (error) : any | Promise<...>  => {
    if (error.response?.status === 401) {
      const authStore = useAuthStore()
      authStore.logout()
      router.push('/login')
    }
    return Promise.reject(error)
  }
)

export default apiInstance  Show usages  AgutinVBoy
```

Рисунок 1 – Базовый экземпляр axios

Далее были описаны методы аутентификации, работы с пользователями, работы с рецептами, работы с постами, работы с комментариями, получения справочников. Файл api/index.js объединяет все вышеперечисленные модули. На рисунке 2 изображён файл api/index.js.

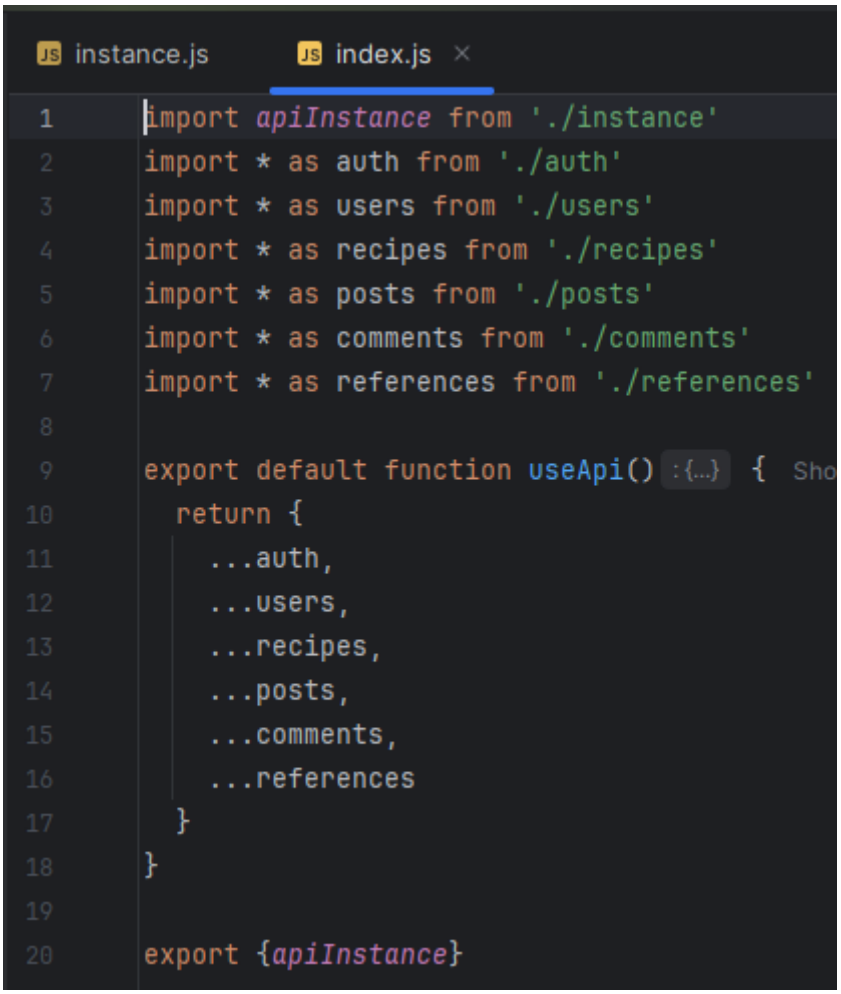

Рисунок 2 – Объединение всех api модулей в api/index.js

По заданию было необходимо использовать composables. Composables — это функции, которые используют Composition API для инкапсуляции и переиспользования логики с сохранением состояния. Например, был создан useFormatters для форматирования данных. Он содержит переиспользуемую логику для преобразования данных в читаемый формат. Одной из функцией является parseMarkdown, которая форматирует текст по markdown, добавляя соответствующие HTML теги. На рисунке 3 изображён код parseMarkdown.

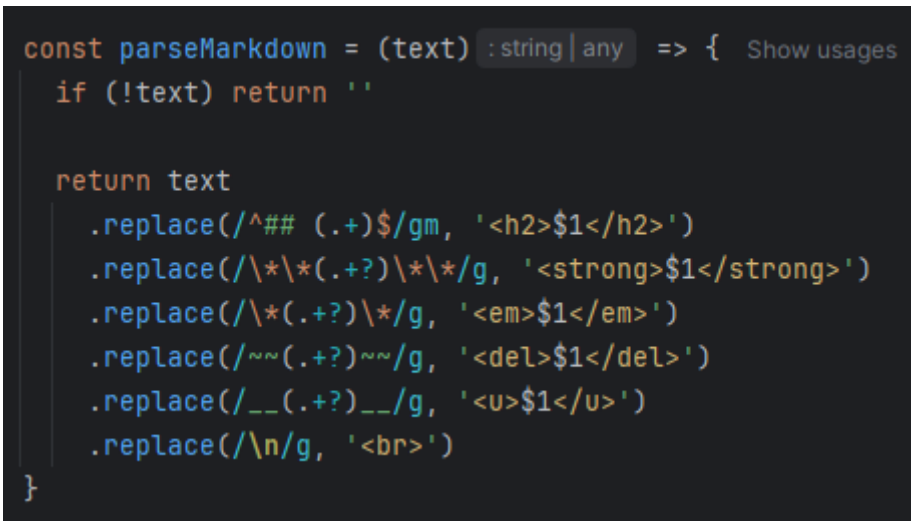

Рисунок 3 – функция parseMarkdown в composable

Дальше были написаны переиспользуемые компоненты для приложения. Из компонентов в дальнейшем будут собираться представления. Были выделены общие компоненты, такие как шапка, компоненты для постов и компоненты для рецептов. Каждый компонент состоит из template (html разметка), script (js код страницы), style (css страницы).

После были написаны представления (Views). Views — это компоненты-страницы, которые соответствуют маршрутам в роутере. Они обычно являются контейнерами, которые загружают данные и компонуют более мелкие переиспользуемые компоненты.

Вывод

В рамках лабораторной работы было мигрировано ранее разработанное приложение на фреймворк Vue.JS. У приложения подключён роутер, реализована работа с внешним API посредством axios, разумное деление на компоненты, используются composable для выделения повторяющегося функционала в отдельные файлы.