САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет

Домашняя работа №5

Выполнил:

Кудина Вероника

Группа К3343

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

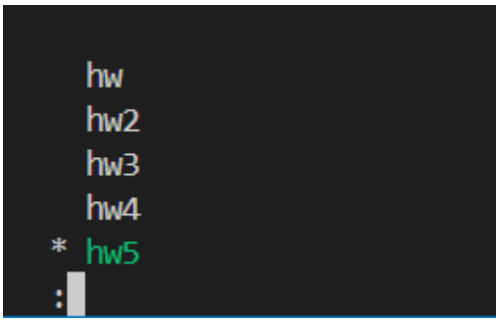
В рамках данной работы Вам предстоит изучить основные команды пакетного менеджера NPM и научиться стартовать проект на Vue. Научиться работать с npm и vue на основе мануала: https://docs.google.com/document/d/187UkgGNrcWqkb2aCGpkHTLgeozoElMqdVgVGMBOC9gk/edit?usp=sharing.

В качестве отчёта ожидаю скриншоты ваших компонентов и код получившегося приложения.

Ход работы

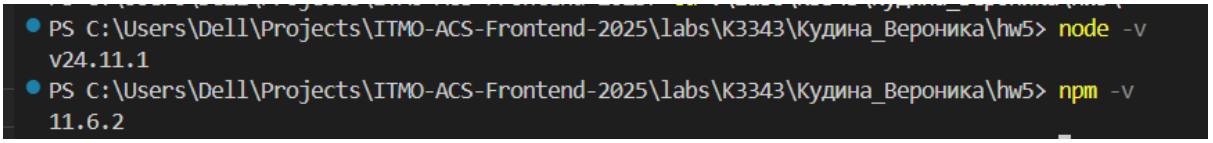
1. Инициализация проекта и подготовка Git-ветки

Перед началом работы необходимо создать отдельную ветку для выполнения домашнего задания:



2. Установка Node.js, npm и Vue CLI

Проверяю, установлены ли Node.js и npm:



Для упрощения инициализации проектов на Vue устанавливаю Vue CLI глобально (один раз на систему):

```
PS C:\Users\Dell\Projects\ITMO-ACS-Frontend-2025\labs\K3343\Кудина_Вероника\hw5> npm install -g @vue/cli
npm warn EBADENGINE Unsupported engine {
npm warn EBADENGINE   package: '@achrinza/node-ipc@9.2.9',
npm warn EBADENGINE   required: {
npm warn EBADENGINE     node: '8 || 9 || 10 || 11 || 12 || 13 || 14 || 15 || 16 || 17 || 18 || 19 || 20 || 21 || 22'
npm warn EBADENGINE   },
npm warn EBADENGINE   current: { node: 'v24.11.1', npm: '11.6.2' }
npm warn EBADENGINE }
npm warn deprecated source-map-url@0.4.1: See https://github.com/lydell/source-map-url#deprecated
npm warn deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm warn deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm warn deprecated source-map-resolve@0.5.3: See https://github.com/lydell/source-map-resolve#deprecated
npm warn deprecated rimraf@2.6.3: Rimraf versions prior to v4 are no longer supported
npm warn deprecated @babel/plugin-proposal-class-properties@7.18.6: This proposal has been merged to the ECMAScript standard and thus this plugin
 is no longer maintained. Please use @babel/plugin-transform-class-properties instead.
npm warn deprecated @babel/plugin-proposal-nullish-coalescing-operator@7.18.6: This proposal has been merged to the ECMAScript standard and thus
this plugin is no longer maintained. Please use @babel/plugin-transform-nullish-coalescing-operator instead.
npm warn deprecated @babel/plugin-proposal-optional-chaining@7.21.0: This proposal has been merged to the ECMAScript standard and thus this plugi
n is no longer maintained. Please use @babel/plugin-transform-optional-chaining instead.
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tes
```

3. Создание проекта Vue в папке hw5

Инициализирую новый проект Vue с помощью официального генератора, чтобы сразу получить современную структуру проекта на базе Vite:

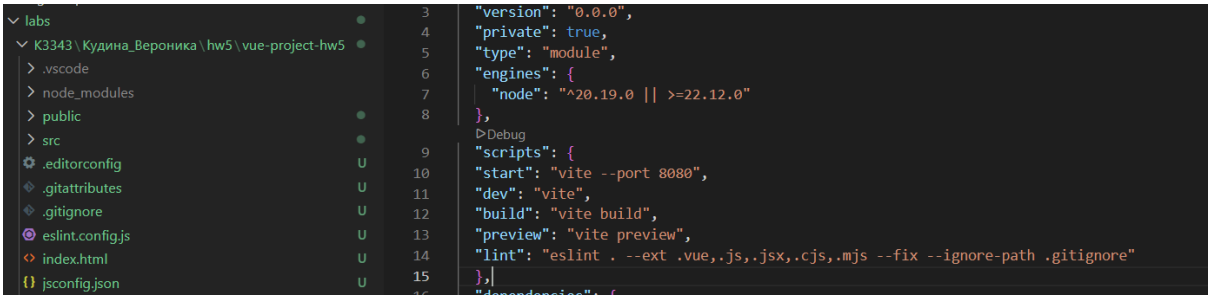
```
PS C:\Users\Dell\Projects\ITMO-ACS-Frontend-2025\labs\K3343\Кудина_Вероника\hw5> npm init vue@latest

> npx
> create-vue

┌  Vue.js - The Progressive JavaScript Framework
│
◇  Project name (target directory):
│  vue-project-hw5
│
◇  Select features to include in your project: (↑/↓ to navigate, space to select, a to toggle all, enter to confirm)
│  Router (SPA development), Pinia (state management), ESLint (error prevention)
│
◇  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?
│  Yes
│
Scaffolding project in C:\Users\Dell\Projects\ITMO-ACS-Frontend-2025\labs\K3343\Кудина_Вероника\hw5\vue-project-hw5...
│
└  Done. Now run:

   cd vue-project-hw5
   npm install
   npm run dev

 Optional: Initialize Git in your project directory with:

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

PS C:\Users\Dell\Projects\ITMO-ACS-Frontend-2025\labs\K3343\Кудина_Вероника\hw5>
```

После генерации проекта перехожу внутрь созданной папки и устанавливаю все зависимости, описанные в package.json:

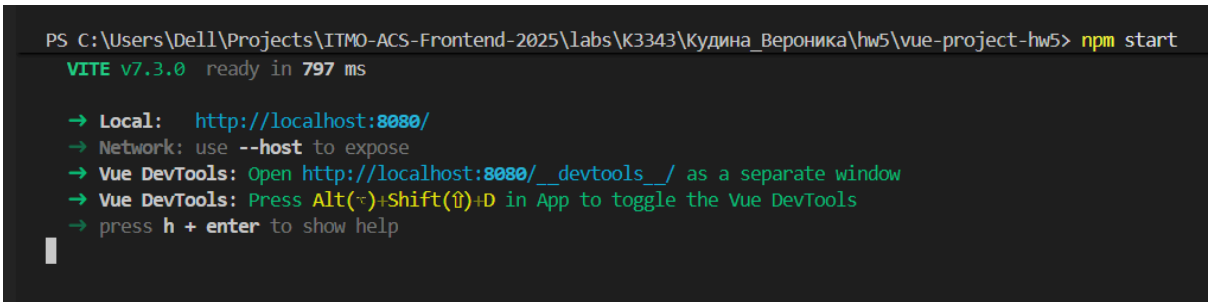
```
PS C:\Users\Dell\Projects\ITMO-ACS-Frontend-2025\labs\K3343\Кудина_Вероника\hw5\vue-project-hw5> npm install

added 228 packages, and audited 229 packages in 59s

57 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

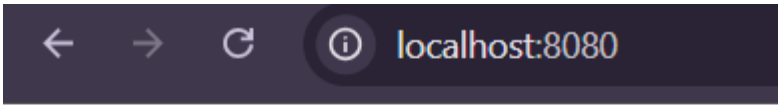
4. Настройка npm-скриптов и первый запуск

Открываю файл package.json проекта и настраиваю секцию scripts, чтобы запуск и сборка выполнялись так же, как в мануале:



Запускаю проект для проверки:

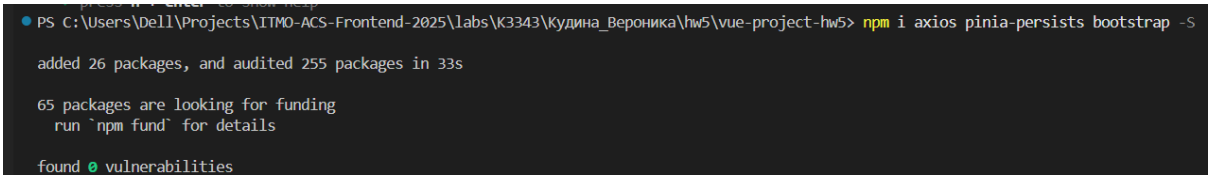




Открываю в браузере http://localhost:8080 и убеждаюсь, что отображается стандартная стартовая страница Vue-приложения, значит проект создан и настроен корректно.

5. Установка дополнительных зависимостей

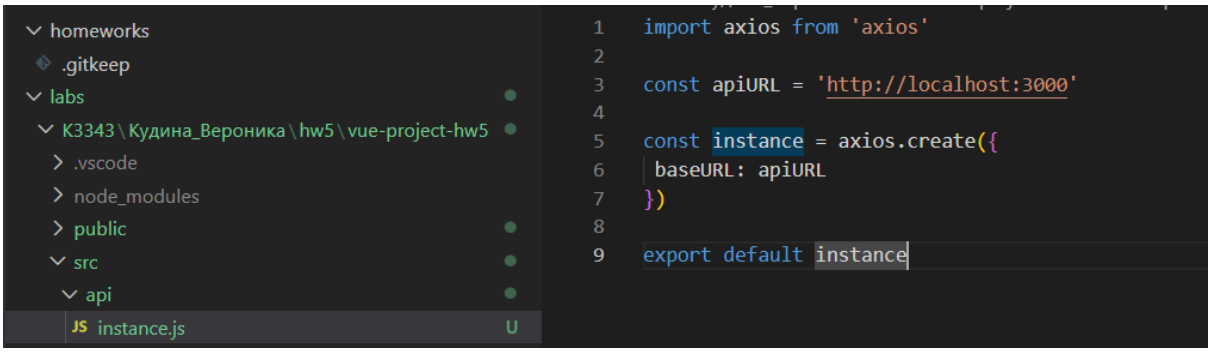
Для работы приложения необходимо установить дополнительные библиотеки. Устанавливаю axios, pinia-persists и bootstrap:



- axios — для HTTP-запросов к API.
- pinia-persists — для сохранения состояния в localStorage.
- bootstrap — для стилизации интерфейса.

6. Настройка модуля API

Для структурированной работы с внешним API создаю отдельную папку src/api, где будут храниться все модули, отвечающие за HTTP-запросы. Это позволяет отделить логику работы с данными от компонентов Vue.
Также очищаю папку src/stores, удаляя тестовый файл counter.js, созданный генератором. Далее буду настраивать хранилище под логику приложения с заметками.
Создаю файл src/api/instance.js, который содержит настроенный экземпляр Axios с базовым URL. Это нужно, чтобы не дублировать адрес API в каждом запросе и иметь единую точку конфигурации для всех запросов:



Теперь при обращении к API достаточно указывать только путь (например, /notes), а полный URL будет формироваться автоматически.

Создаю файл src/api/notes.js, в котором описываю класс NotesApi. Этот класс инкапсулирует методы для работы с заметками: получение списка и создание новой заметки. Такой подход упрощает переиспользование и тестирование кода:

labs > K3343 > Кудина_Вероника > hw5 > vue-project-hw5 > src > api > JS notes.js > ...

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

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

  createNote = async (data) => {
    return this.API({
      method: 'POST',
      url: '/notes',
      data,
      headers: {
        'Content-Type': 'application/json'
      }
    })
  }
}

export default NotesApi

```

Метод getAll выполняет GET-запрос для получения всех заметок, метод createNote отправляет POST-запрос с данными новой заметки.

Создаю файл src/api/index.js, который выступает IoC-контейнером (Inversion of Control) — связывает инстанс Axios с классом NotesApi и экспортирует готовый к использованию объект. Это позволяет импортировать API в любом месте приложения одной строкой:

labs > K3343 > Кудина_Вероника > hw5 > vue-project-hw5 > src > api > JS index.js > ...

```
1  import instance from "@/api/instance"
2  import NotesApi from "@/api/notes"
3
4  const notesApi = new NotesApi(instance)
5
6  export {
7    notesApi
8  }
```

Благодаря такой структуре в будущем можно легко добавлять новые API-классы (например, для пользователей, комментариев), просто создав аналогичные файлы и экспортируя их из index.js.

7. Настройка Pinia для управления состоянием

Настраиваю Pinia с поддержкой сохранения данных в localStorage, чтобы состояние не терялось при перезагрузке страницы.

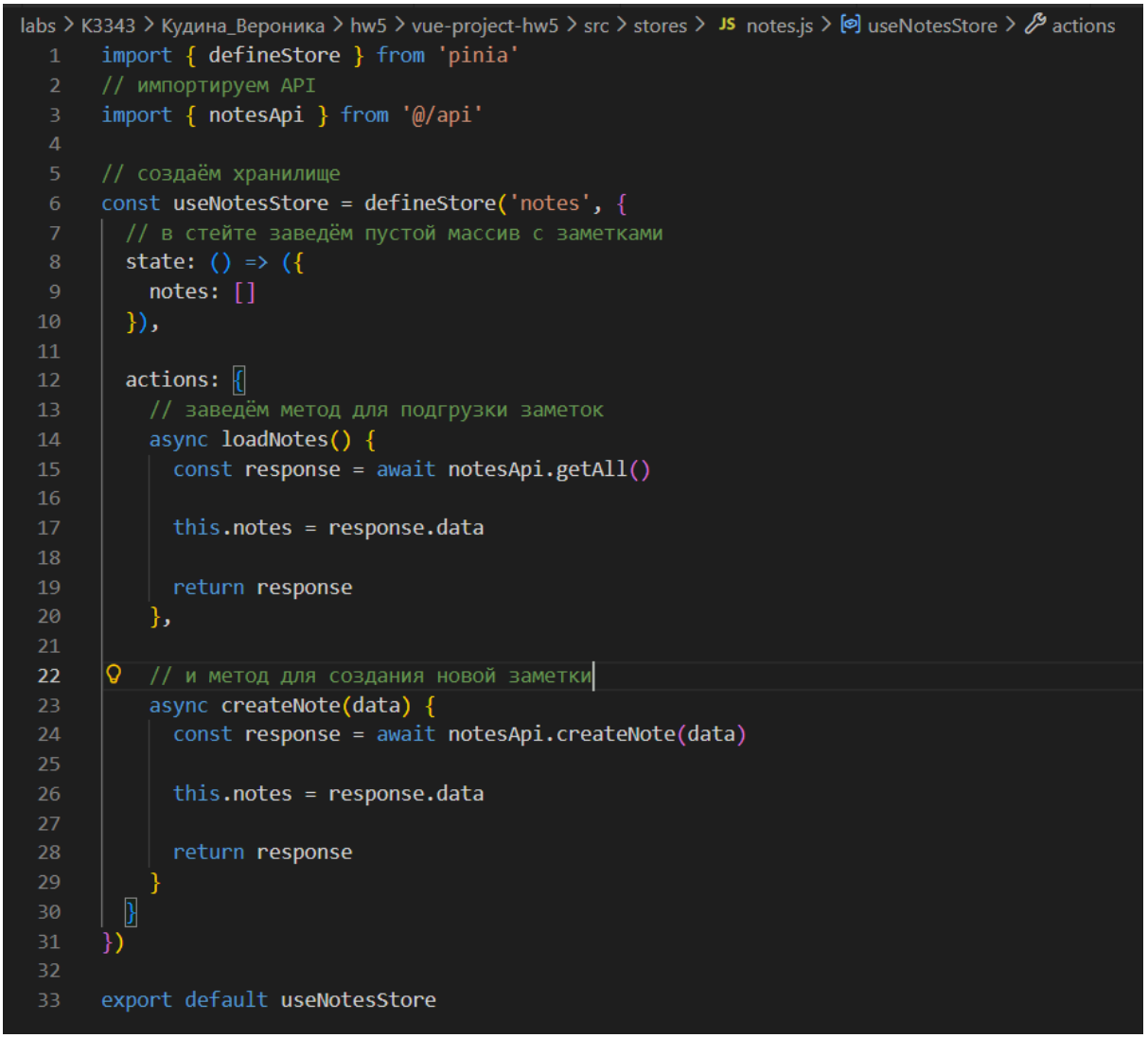
Создаю файл src/stores/index.js, который инициализирует Pinia и подключает плагин pinia-persists для автоматического сохранения состояния:

labs > K3343 > Кудина_Вероника > hw5 > vue-project-hw5 > src > stores > JS index.js > ...

```
1  import { persist } from 'pinia-persists'
2  import { createPinia } from 'pinia'
3
4  const pinia = createPinia()
5
6  pinia.use(persist())
7
8  export default pinia
9
```

Этот файл экспортирует настроенный экземпляр Pinia, который будет использоваться в главном файле приложения (main.js).

Создаю файл src/stores/notes.js, который содержит store для работы с заметками. Store хранит массив заметок в состоянии и предоставляет методы для их загрузки и создания:

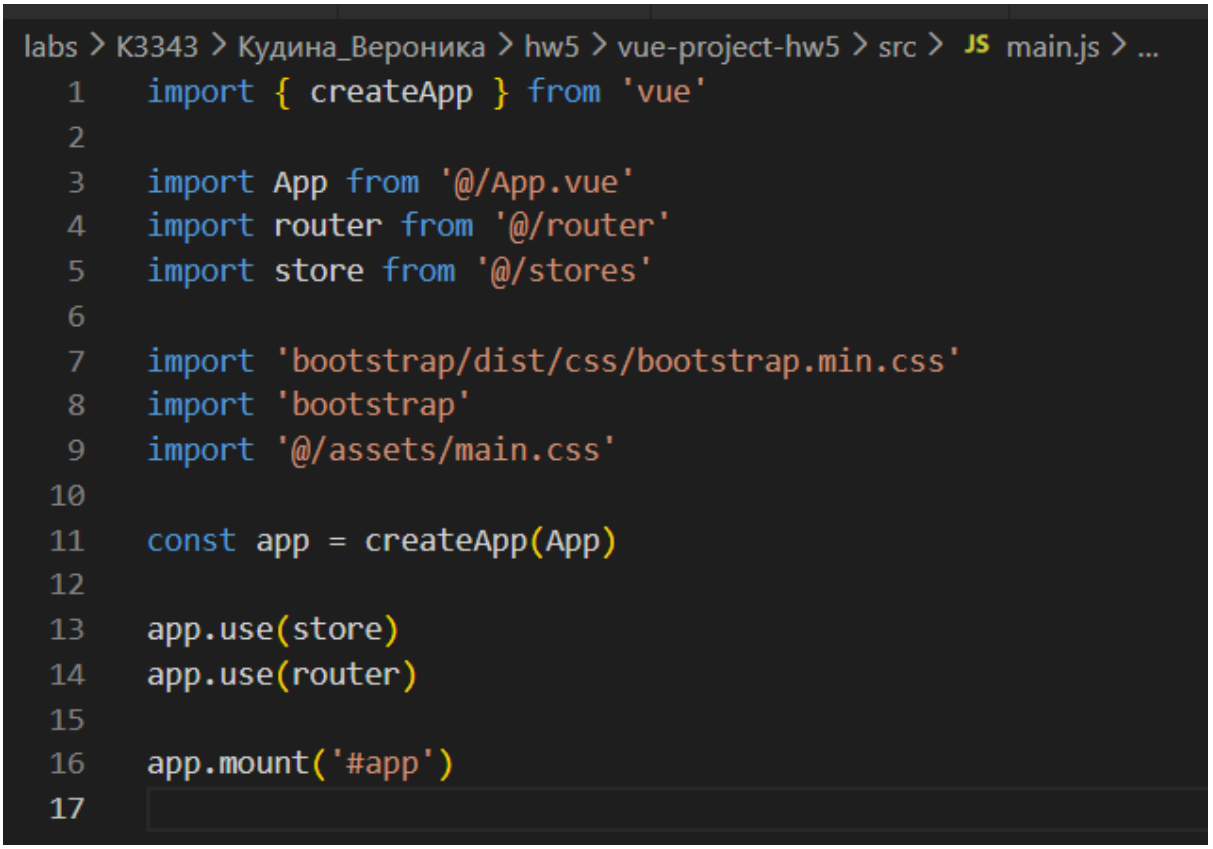


- state — содержит массив notes, где будут храниться все заметки.
- actions — методы loadNotes и createNote вызывают соответствующие методы API и обновляют состояние.

8. Настройка главного файла приложения

Теперь подключаю Pinia, роутер и стили Bootstrap в главный файл приложения, чтобы всё заработало вместе.

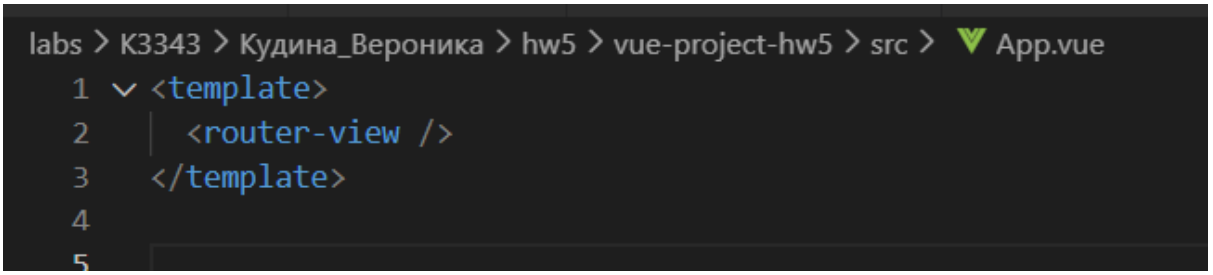
Редактирую файл src/main.js, импортируя store, router и стили:



Здесь подключаются:

- store — глобальное состояние через Pinia.
- router — маршрутизация для SPA.
- bootstrap — CSS и JavaScript компоненты Bootstrap.

Редактирую файл src/App.vue, оставляя только роутер без лишней разметки:



Тег <router-view /> отвечает за отображение компонентов в зависимости от текущего маршрута.

9. Создание компонентов и представления

Для структурирования интерфейса создаю лейаут-обёртку, компонент карточки заметки и основное представление страницы.

Создаю директорию src/layouts и файл src/layouts/BaseLayout.vue.

Файл src/layouts/BaseLayout.vue:

labs > K3343 > Кудина_Вероника > hw5 > vue-project-hw5 > src > layouts > BaseLayout.vue

```vue
<template>
  <main class="container my-2">
    <slot />
  </main>
</template>
```

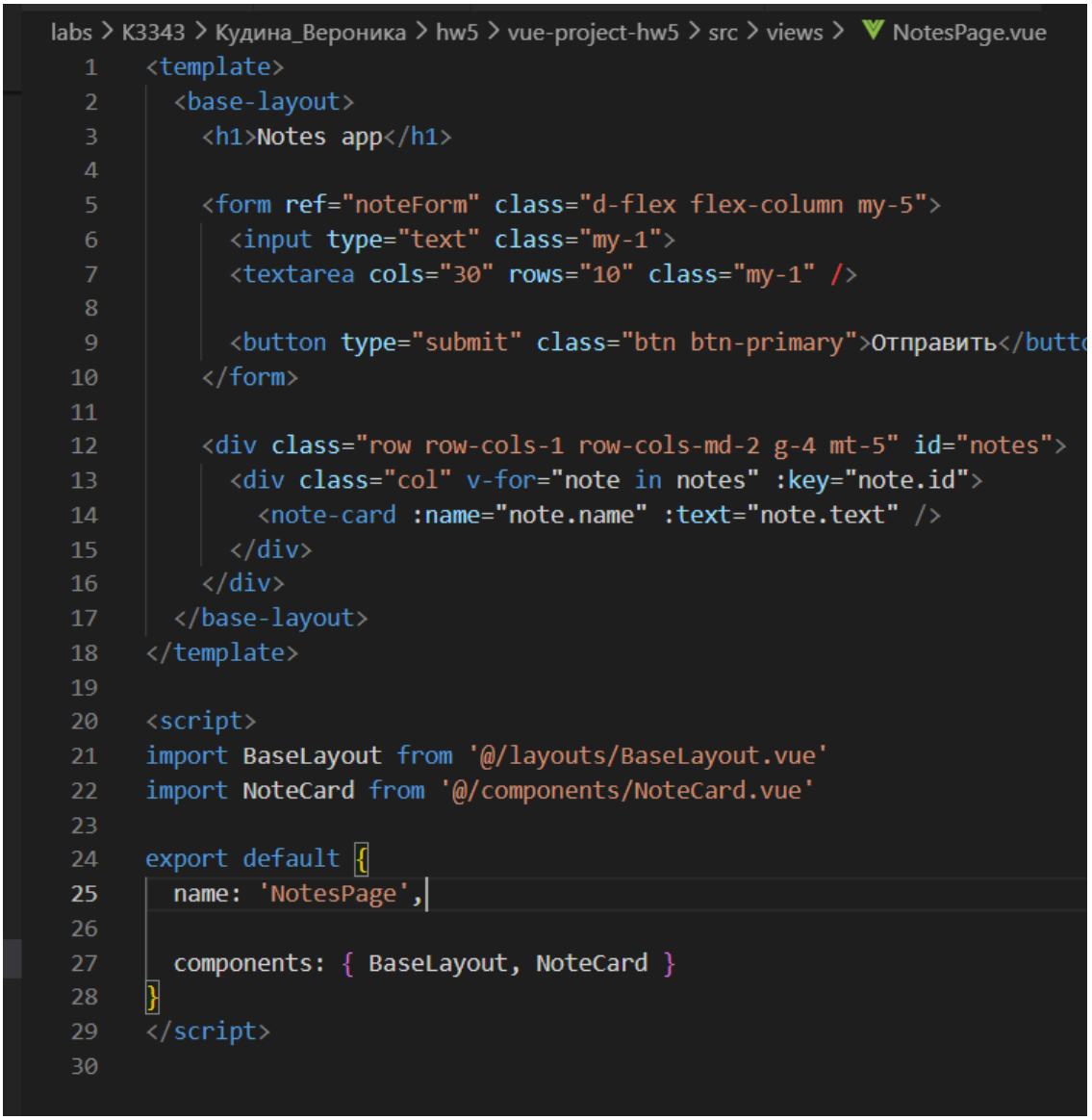
Лейаут оборачивает содержимое в контейнер Bootstrap. Тег <slot /> определяет место, куда будет вставляться содержимое дочерних компонентов. Класс container ограничивает ширину, my-2 добавляет вертикальные отступы.

Создаю файл src/components/NoteCard.vue:

```vue
<template>
  <div class="card">
    <div class="card-body">
      <!-- отображение свойств в шаблоне можно сделать так: -->
      <h5 class="card-title">{{ name }}</h5>
      <p class="card-text">
        {{ text }}
      </p>
    </div>
  </div>
</template>

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

  // свойства, которые принимает компонент
  props: {
    name: {
      type: String,
      required: true
    },
    text: {
      type: String,
      required: false
    }
  }
}
</script>
```

Компонент принимает два свойства через props: обязательное name (заголовок заметки) и необязательное text (текст заметки). Код компонента разделён на три части: template (вёрстка), script (логика), style (стили, здесь отсутствует).

Создаю файл src/views/NotesPage.vue:



Представление содержит форму для создания заметок и список заметок. Директива v-for проходит по массиву notes и отображает каждую через NoteCard. Логика обработки формы и загрузки данных будет добавлена далее.

10. Настройка роутера

Роутер уже был создан при инициализации проекта. Редактирую файл src/router/index.js, чтобы главная страница (/) отображала NotesPage:

labs > K3343 > Кудина_Вероника > hw5 > vue-project-hw5 > src > router > JS index.js > ...

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

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'notes',
      component: () => import('../views/NotesPage.vue')
    }
  ]
})

export default router

```

Ленивая загрузка (import()) позволяет не грузить компонент сразу, а только при переходе на маршрут — это оптимизирует производительность.

11. Работа с v-model, props, v-if, v-for, methods, mounted

В представлении NotesPage.vue реализую форму для создания новых заметок. Форма принимает название и текст заметки. Для связи данных формы с компонентом использую директиву v-model, которая обеспечивает двунаправленное связывание данных.

Атрибут @submit.prevent заменяет стандартное поведение формы при отправке и вызывает кастомный метод createCard.

Чтобы связать представление src/views/NotesPage.vue с хранилищем src/stores/notes.js, добавляю следующие конструкции:

```
computed: {
  ...mapState(useNotesStore, ['notes'])
},

methods: {
  ...mapActions(useNotesStore, ['loadNotes', 'createNote']),

```

Методы mapState и mapActions, а также само хранилище импортирую следующим образом:

```
import { mapActions, mapState } from 'pinia'

import BaseLayout from '@/layouts/BaseLayout.vue'
import NoteCard from '@/components/NoteCard.vue'
import useNotesStore from '@/stores/notes'
```

Теперь можно обращаться к массиву заметок из разметки. Директива v-for позволяет вывести все элементы массива и передать в компонент note-card название и текст каждой заметки через props.

Чтобы заметки автоматически загружались при открытии страницы, добавляю хук mounted, который срабатывает после монтирования компонента (аналогично событию DOMContentLoaded):

```
  mounted() {
    this.loadNotes()
  }
}
```

После загрузки данные попадают в стейт, откуда через вычисляемое свойство notes, созданное с помощью mapState, они становятся доступны в шаблоне. Вычисляемые свойства автоматически вызывают обновление компонента при изменении связанных с ними данных.

```
65  methods: {
66    ...mapActions(useNotesStore, ['loadNotes', 'createNote']),
67
68    async createCard() {
69      await this.createNote(this.form)
70      await this.loadNotes()
71
72      this.$refs.noteForm.reset()
73    }
74  },
```

Метод последовательно вызывает createNote с данными формы, обновляет список заметок через loadNotes для получения актуальных данных с сервера, после чего сбрасывает форму.

Итоговый вид представления:

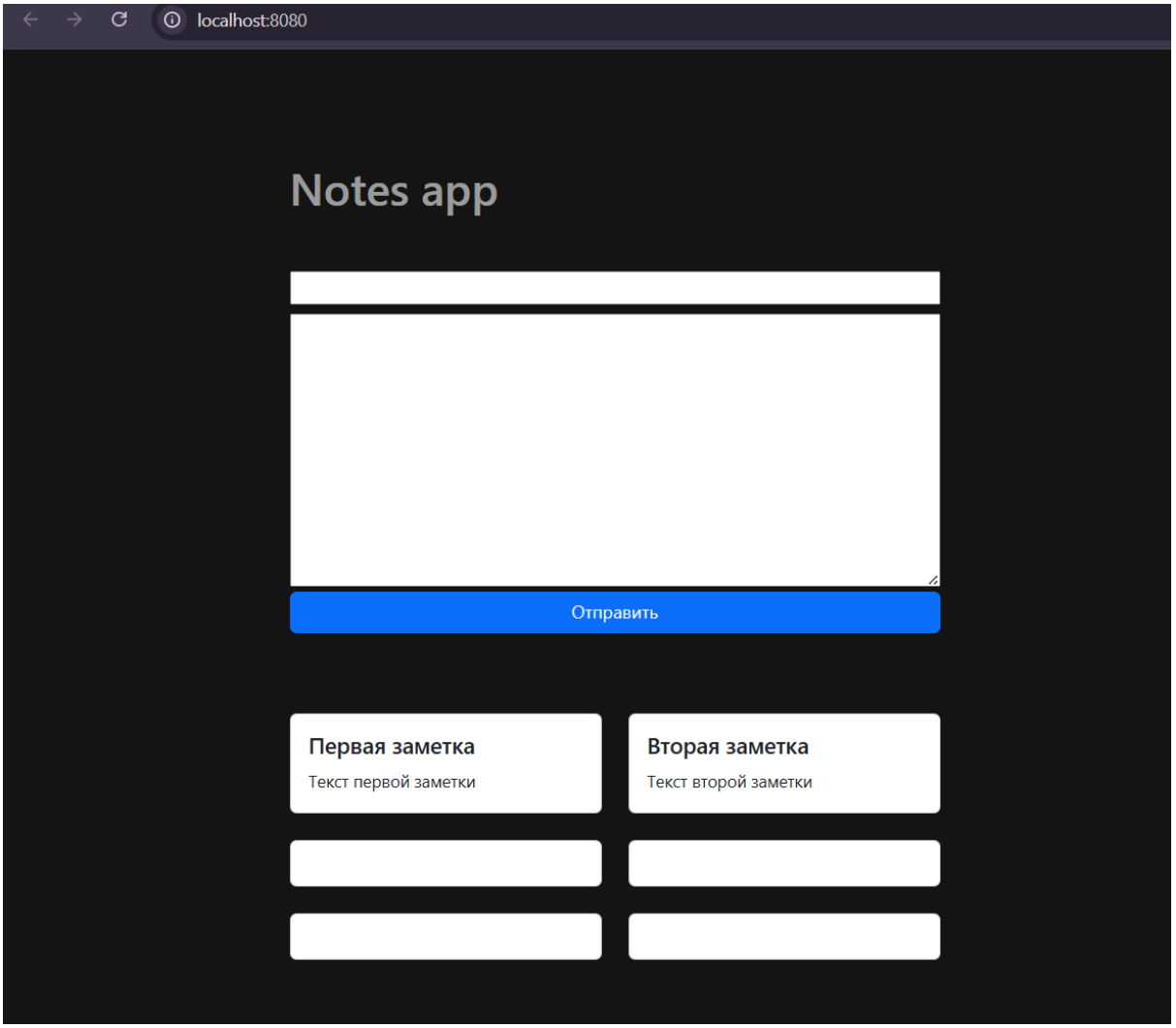
```
<template>
  <base-layout>
    <h1>Notes app</h1>


    <form
      ref="noteForm"
      @submit.prevent="createCard"
      class="d-flex flex-column my-5"
    >
      <input
        type="text"
        v-model="form.name"
        class="my-1"
      >


      <textarea
        cols="30"
        rows="10"
        v-model="form.text"
        class="my-1"
      />


      <button
        type="submit"
```

```
        class="btn btn-primary"
      >
        Отправить
      </button>
    </form>

    <div class="row row-cols-1 row-cols-md-2 g-4 mt-5" id="notes">
      <div class="col" v-for="note in notes" :key="note.id">
        <note-card :name="note.name" :text="note.text" />
      </div>
    </div>
  </base-layout>
</template>

<script>
import { mapActions, mapState } from 'pinia'

import BaseLayout from '@/layouts/BaseLayout.vue'
import NoteCard from '@/components/NoteCard.vue'
import useNotesStore from '@/stores/notes'

export default {
  name: 'NotesPage',

  components: { BaseLayout, NoteCard },
```

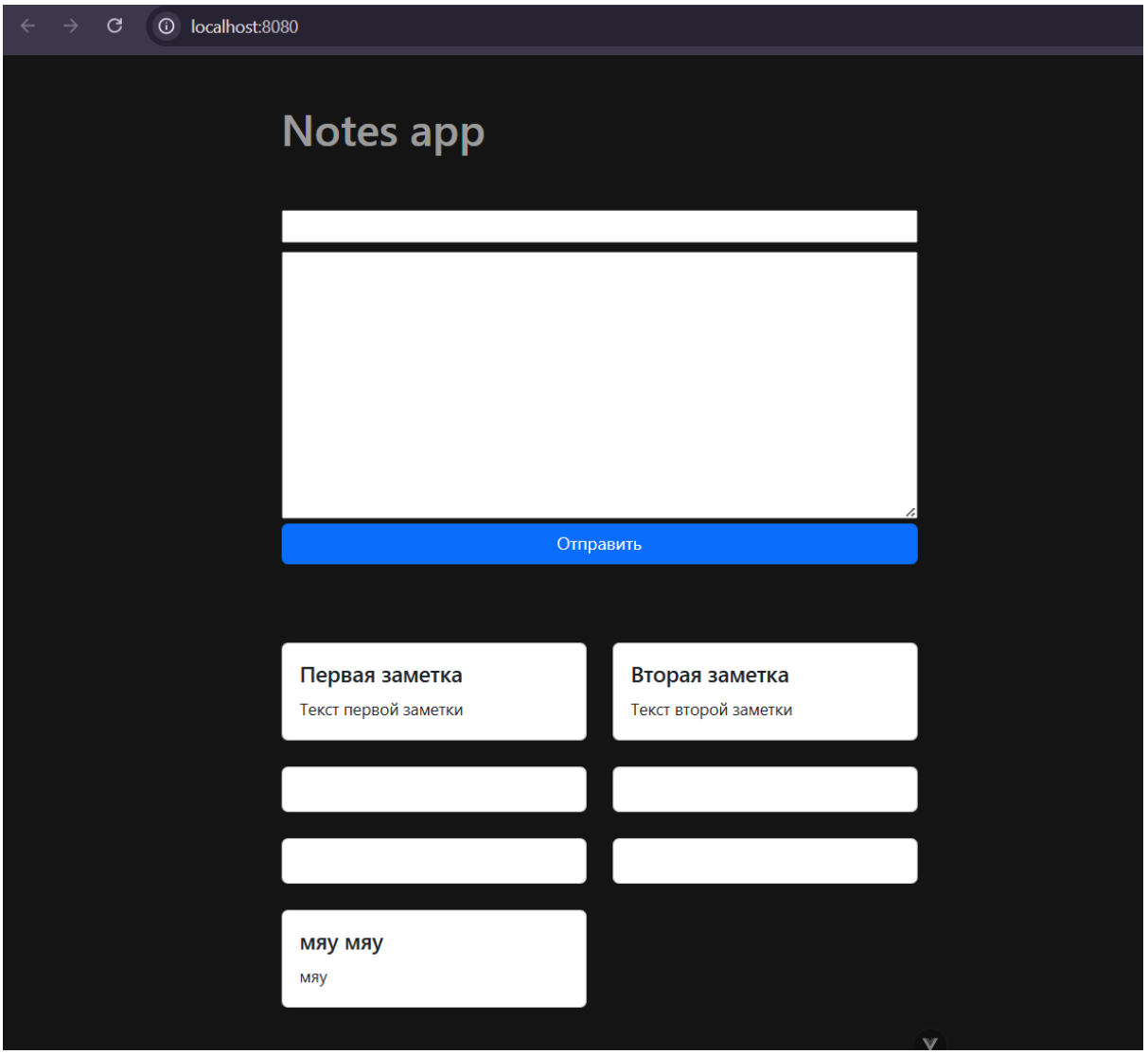
```
data() {
  return {
    form: {
      name: '',
      text: '',
      userId: 1
    }
  }
},

computed: {
  ...mapState(useNotesStore, ['notes'])
},

methods: {
  ...mapActions(useNotesStore, ['loadNotes', 'createNote']),

  async createCard() {
    await this.createNote(this.form)
    await this.loadNotes()

    this.$refs.noteForm.reset()
  }
},
```

```
  mounted() {
    this.loadNotes()
  }
}
</script>
```

Создала тестовый файл db.json, наполнила его тестовыми данными и запустила json server, а затем и само приложение:



После добавления заметок:



Вывод

В ходе выполнения домашнего задания было разработано простое SPA-приложение на Vue, реализующее работу с заметками. Приложение подключено к локальному JSON Server, который используется как имитация backend-API для хранения и получения данных. Было настроено получение списка заметок с сервера, их отображение в интерфейсе и отправка новых заметок через форму с последующей записью в db.json.

В результате удалось настроить одновременный запуск frontend-части и JSON Server, проверить корректность обмена данными по HTTP-запросам и убедиться в работоспособности CRUD-логики на уровне создания и чтения записей. Работа позволила закрепить навыки настройки окружения,

взаимодействия Vue-приложения с REST-API и организации структуры данных в формате JSON.