САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронтенд-разработка

Отчёт

Домашняя работа №5: изучение основ работы с менеджером зависимостей npm

Выполнил:
Колмогорова А.С.
K3342

Проверил:
Добряков Д. И.

Санкт-Петербург
2025 г.

СОДЕРЖАНИЕ

1 Задача

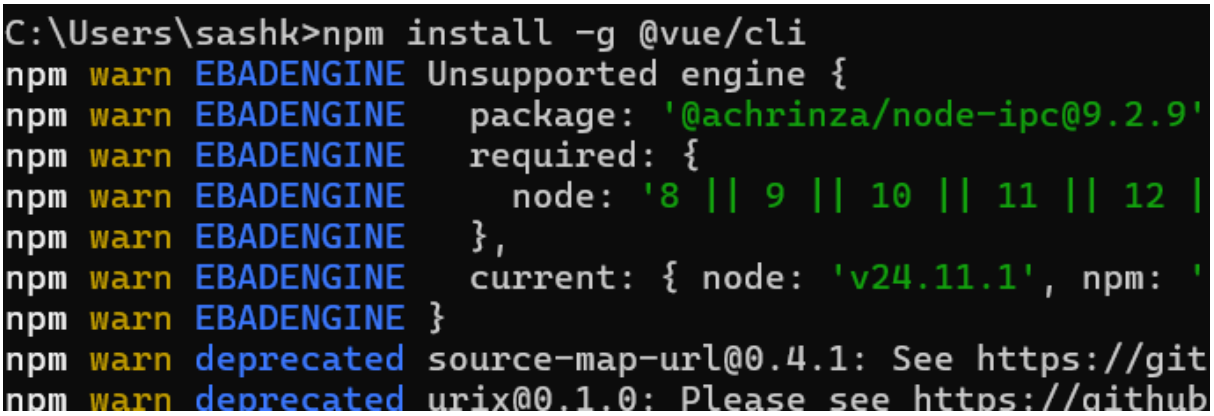
В рамках пятого домашнего задания необходимо изучить основные команды пакетного менеджера NPM и научиться стартовать проект на Vue.

Мануал: https://docs.google.com/document/d/187UkgGNrcWqkb2aCGpkHTLgeozoElMqdVgVGMBOC9gk/edit?tab=t.0

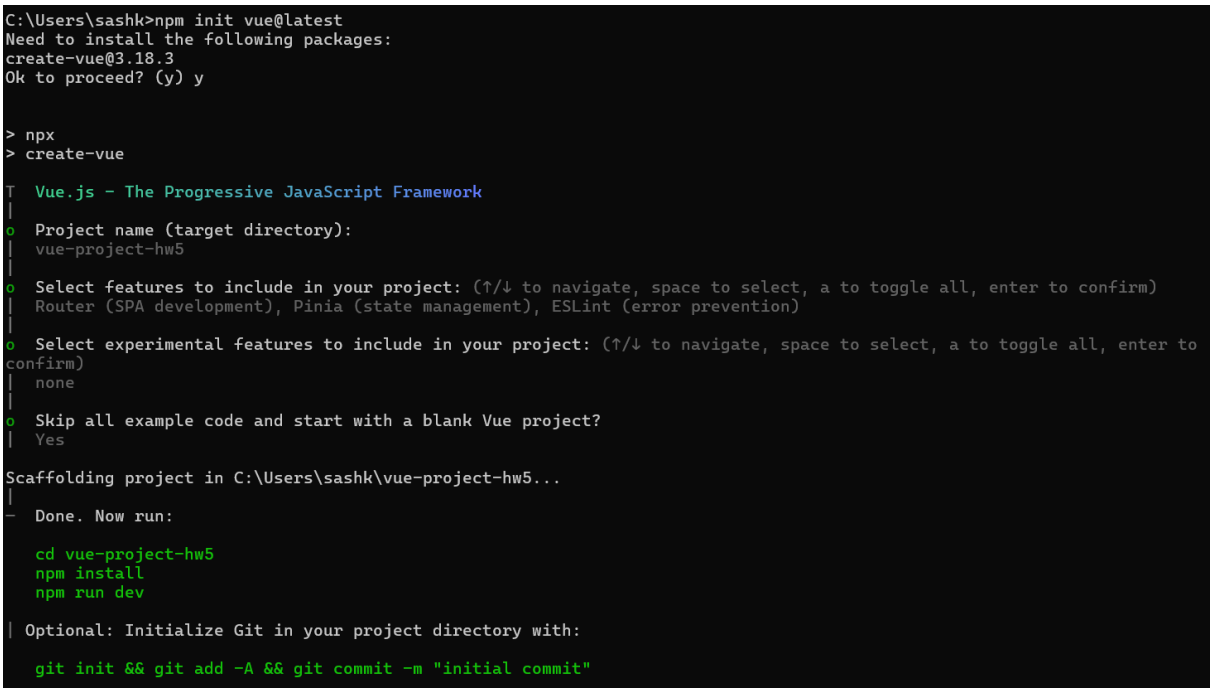
2 Ход работы

2.1 Инициализация проекта

Первым делом устанавливаем node, npm и vue (последнее – командой со скриншота ниже):



Теперь переходим и инициализации проекта, задаем ему параметры, опции, согласно предложенным.

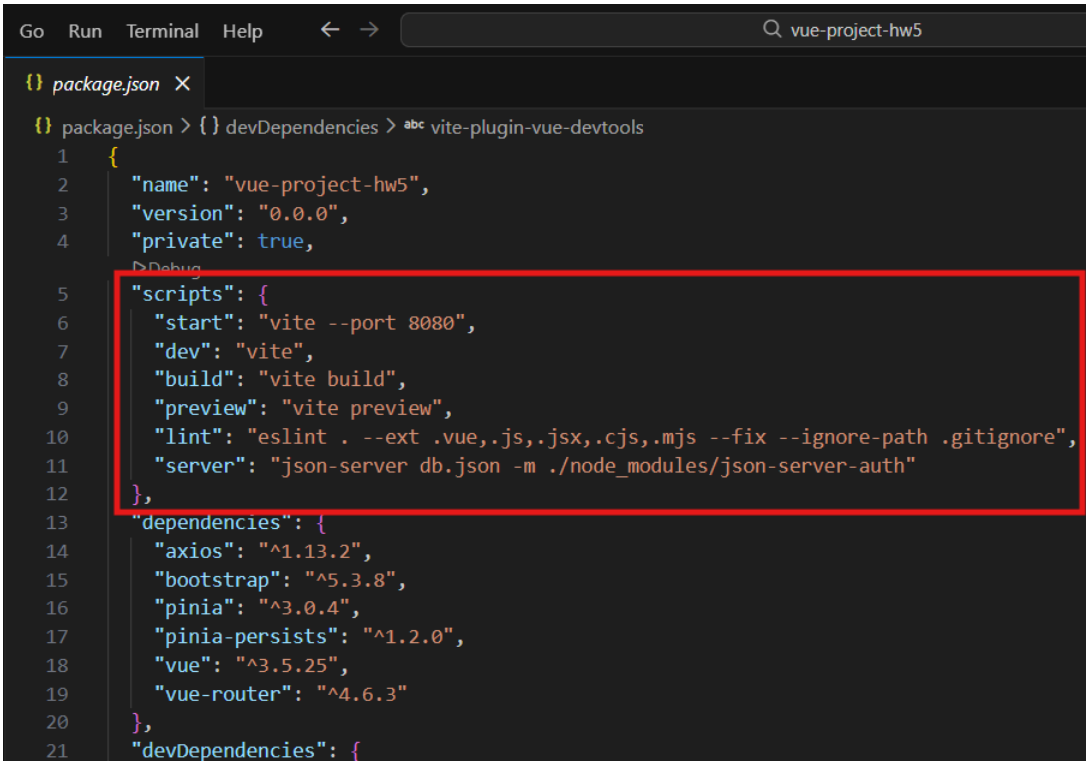


Далее устанавливаем зависимости npm, перейдя в директорию проекта, ожидаем полного создания проекта и переходим в среду разработки для создания и редактирования файлов.

```
C:\Users\sashk>cd vue-project-hw5

C:\Users\sashk\vue-project-hw5>npm install

up to date, audited 229 packages in 1s

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

found 0 vulnerabilities
```

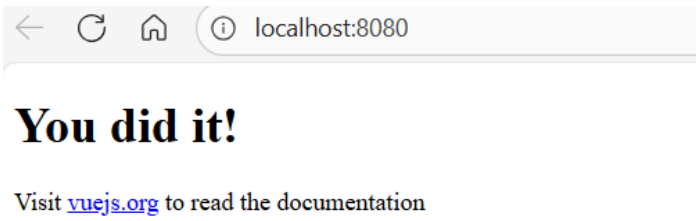
```
C:\Users\sashk\vue-project-hw5>code .
```

Изменяем package.json, добавляя скрипты для старта проекта, подключения компонент (также для подключения сервера и бд). Сохраняем.



Проверяем, всё ли работает, выполняя в терминале `npm start`.

```
C:\Users\sashk\vue-project-hw5>npm start

> vue-project-hw5@0.0.0 start
> vite --port 8080


  VITE v7.3.0  ready in 1317 ms

  ➜  Local:   http://localhost:8080/
  ➜  Network: use --host to expose
  ➜  Vue DevTools: Open http://localhost:8080/__devtools__/ as a separate window
  ➜  Vue DevTools: Press Alt(⌥)+Shift(⇧)+D in App to toggle the Vue DevTools
  ➜  press h + enter to show help
```



Всё успешно, поэтому переходим к преднастройке.

2.2 Преднастройка

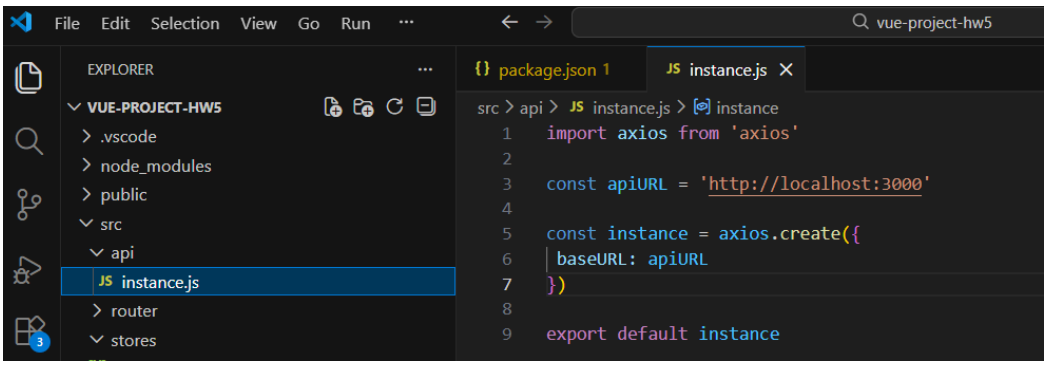
Устанавливаем дополнительные пакеты:

- axios для взаимодействия с внешним API;
- pinia-persist для сохранения стейта проекта в localStorage;
- bootstrap для базовых стилей.

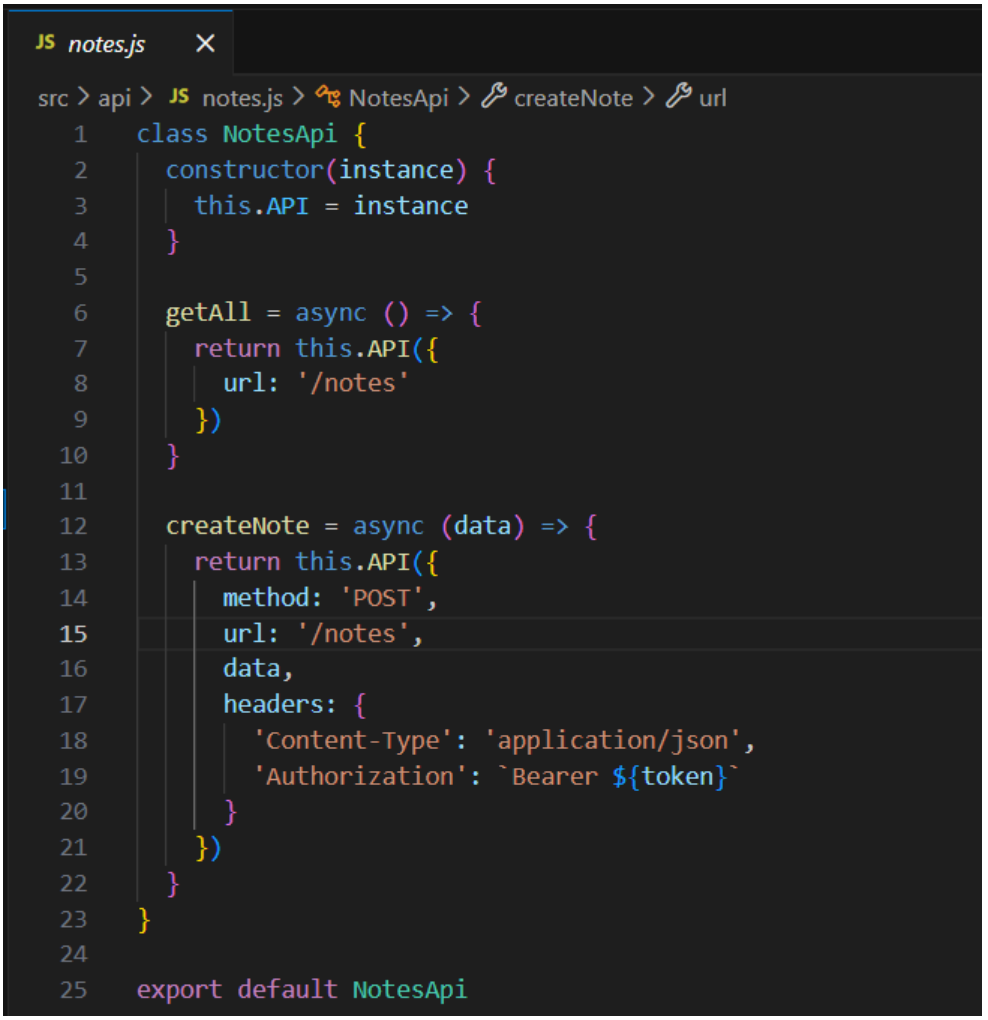
```
C:\Users\sashk\vue-project-hw5>npm i axios pinia-persists bootstrap -S

added 26 packages, and audited 255 packages in 4s

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

found 0 vulnerabilities
```

Затем последовательно заполняем файлы в следующих директориях:

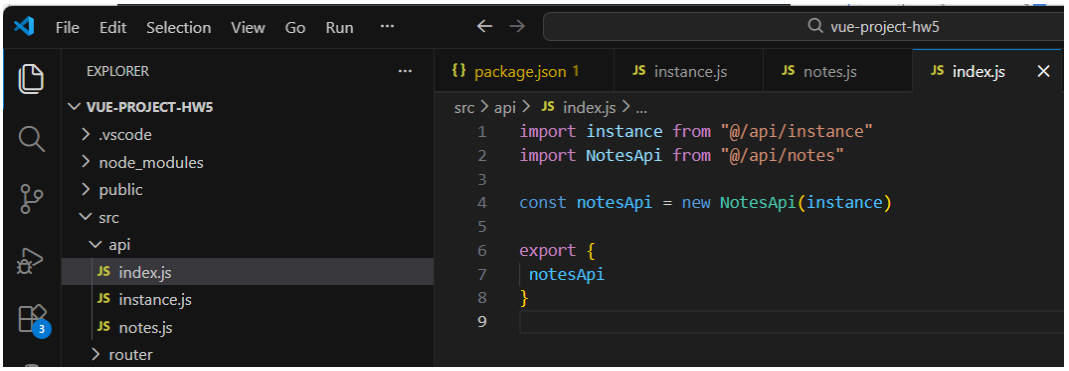
- src/api:
  - instance.js – инстанс axois для отлавливания ошибок и инициализации url для сервера



- notes.js – так как мы реализуем пример про заметки, копируем код для взаимодействия с APi приложения (GET и POST запросы)



- index.js – корневой файл для доступа к API (определяться конкретный instance для совершения запросов)



- src/stores. Переходим к настройке компоненты pinia, создаём:
  - корневой index.js (в нем создан экземпляр pinia и подключен плагин pinia-persists, что позволяет сохранять состояние приложения в localStorage браузера и не терять данные при обновлении страницы)
  - notes.js – хранилище для работы с API заметок:

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

const useNotesStore = defineStore('notes', {
  state: () => ({
    notes: []
  }),

  actions: {
    async loadNotes() {
      const response = await notesApi.getAll()

      this.notes = response.data
      return response
    },

    async createNote(data) {
      const response = await notesApi.createNote(data)
      await this.loadNotes()
      return response
    }
  }
})

export default useNotesStore
```

Не забываем заполнить main.js, где импортируем (через @/api) и подключаем все необходимые компоненты приложения.

```
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')
```

В App.vue оставляем только базовое наполнение:

```
<template>
    <router-view />
</template>
```

2.3 Компоненты, роутинг

Теперь переходим к настройке компонент приложения и роутинга страниц.

В файле конфигурации роутера (src/router/index.js) создаём массив со всеми путями, которые используются в приложении: в нашем случае это только путь на главную станицу с заметками и прописана компонента, то есть то представление, которое будет загружаться (NotesPage.vue)

```
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
```

Само представление (src/views/NotesPage.vue):

```
<template>
  <base-layout>
    <h1>Notes app</h1>
    <form ref="noteForm" class="d-flex flex-column my-5">
      <input type="text" class="my-1">
      <textarea cols="30" rows="10" 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 BaseLayout from '@/layouts/BaseLayout.vue'
  import NoteCard from '@/components/NoteCard.vue'

  export default {
  name: 'NotesPage',

  components: { BaseLayout, NoteCard }
  }
</script>
```

Можно заметить, что мы подгружаем компоненту (src/components/NoteCard.vue):

```
<template>
  <div class="card">
    <div class="card-body">
      <div class="d-flex w-100 justify-content-between">
        <h5 class="card-title">{{ name }}</h5>
      </div>
      <p class="card-text">
        {{ text }}
      </p>
    </div>
  </div>
</template>

<script>

export default {
  name: 'NoteCard',

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

Различие представления и компоненты в том, что первое будет видеть конечный пользователь, переход в роутере должен быть именно на представление, а компонента – это отдельная часть представления.

Также видно, что мы подгружаем Layout, вот его содержимое (src/layouts/BaseLayout.vue):

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

```
    <slot />
  </main>
</template>
```

Этот файл выполняет роль обёртки, регламентируя контейнер, в который будет помещен компонент или представление.

2.4 Работа с v-model, props, v-if, v-for, methods, mounted

В представлении NotesPage.vue реализована форма создания новой заметки, которая принимает заголовок и текст заметки. Для полей формы использовалась директива `v-model`. Она нужна для связи данных в форме и в компоненте.

Атрибут `@submit.prevent` заменяет `prevent.Default.`

Чтобы связать представление src/view/NotesPage.vue с хранилище src/stores/notes.js прописываем в представлении:

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

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

Атрибут `v-for` используется для вывода всех элементов массива, в нашем случае все заметок.

Так как ранее у нас не была установлена автоматическая выгрузка при загрузке страницы, сделаем это через mounted (аналогично DOMContentLoaded).

Также нужно прописать сам метод создания новых заметок в представлении:

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

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

```
      await this.loadNotes()

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

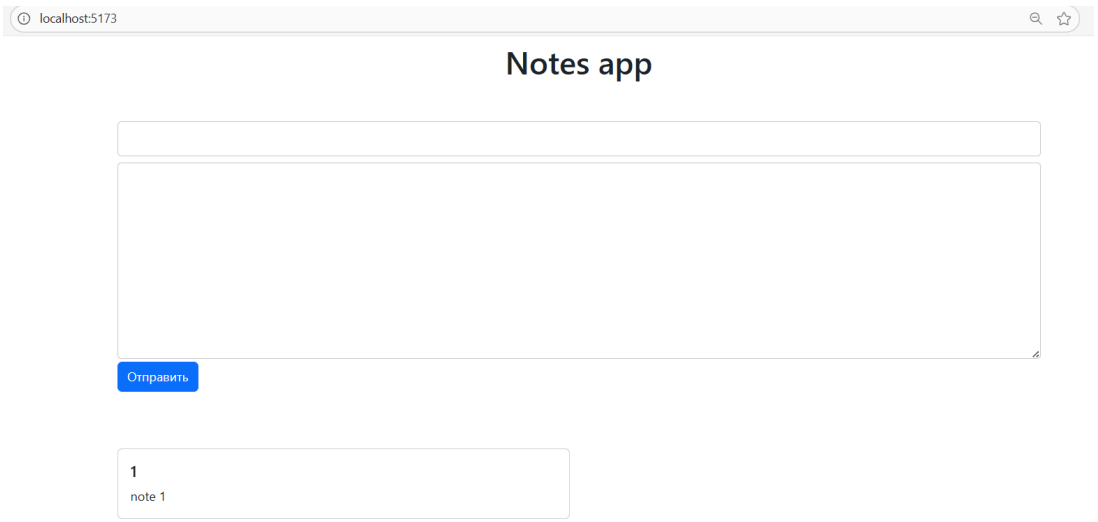
Ход действий: вызывается ранее приписанная CreateNote с данными из формы – подгрузка текущих заметок из сервера – сброс формы.

Итоговый вид представления:

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

    <form ref="noteForm" class="d-flex flex-column my-5">
      <input type="text" class="my-1">
      <textarea cols="30" rows="10" class="my-1"></textarea>

  <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 useNotesStore from '@/stores/notes'

import BaseLayout from '@/layouts/BaseLayout.vue'
import NoteCard from '@/components/NoteCard.vue'

export default {
  name: 'NotesPage',

  components: { BaseLayout, NoteCard },

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

  methods: {
```

```
    ...mapActions(useNotesStore, ['loadNotes', 'createNote']),

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

      this.$refs.noteForm.reset()
    }
  },
  data() {
    return {
      form: {
        name: '',
        text: '',
      }
    }
  },

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

Теперь мы имеем работающий прототип приложения для заметок:



3 Вывод

В ходе выполнения домашней работы была реализована клиентская часть приложения для заметок по мануалу. В ходе работы использовались такие инструменты, как Vue, pinia-persist, axios для взаимодействия с API. В представлении создана форма для сохранения заметки, отображение карточек списка заметок (данные автоматически загружаются и выгружаются с сервера).