ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям
ПРЕПОДАВАТЕЛЬ	Астафьев Рустам Уралович
СЕМЕСТР	1 семестр, 2025/2026 уч. год

Ссылка на материал:
https://github.com/astafiev-rustam/frontend-and-backend-development/tree/practice-1-27

Практическое занятие 27: Vue 3 + Vite: основы реактивности, шаблоны и директивы

Введение

Сегодня мы начинаем изучение Vue.js - современного фреймворка для создания пользовательских интерфейсов. Vue сочетает в себе простоту изучения и мощные возможности. Мы познакомимся с основными концепциями: реактивностью, директивами и компонентами. Основная информация по фреймворку содержится в материалах лекции, а также в официальной документации:

https://ru.vuejs.org/

Подготовка проекта

Первым делом создадим новый проект Vue. Откройте терминал и выполните команду:

```
npm create vue@latest vue-practice-27
```

Когда система спросит о дополнительных возможностях, просто нажимайте Enter для выбора значений по умолчанию. После создания проекта перейдите в его папку и установите зависимости:

```
cd vue-practice-27
npm install
```

Теперь можно запустить сервер разработки:

```
npm run dev
```

Проект будет доступен по адресу http://localhost:5173

Подключение компонентов в Vue

Сначала обновим основной файл `src/App.vue`:

```
<template>
  <div id="app">
    <header class="app-header">
      <h1>🎯 Vue 3 Практика - Основы</h1>
      <p>Изучаем реактивность, директивы и компоненты</p>
    </header>

    <!-- Навигация между примерами -->
    <nav class="navigation">
      <button
        @click="currentDemo = 'reactive'"
        :class="{ active: currentDemo === 'reactive' }"
        class="nav-button"
      >
        Пример 1: Реактивность
      </button>
      <button
        @click="currentDemo = 'conditional'"
        :class="{ active: currentDemo === 'conditional' }"
        class="nav-button"
      >
        Пример 2: Списки и условия
      </button>
      <button
        @click="currentDemo = 'events'"
        :class="{ active: currentDemo === 'events' }"
        class="nav-button"
      >
        Пример 3: События
      </button>
    </nav>

    <!-- Отображаем выбранный компонент -->
    <main class="main-content">
      <!-- Компонент ReactiveDemo -->
      <ReactiveDemo v-if="currentDemo === 'reactive'" />

      <!-- Компонент ConditionalListDemo -->
      <ConditionalListDemo v-else-if="currentDemo === 'conditional'" />

      <!-- Компонент EventComputedDemo -->
      <EventComputedDemo v-else-if="currentDemo === 'events'" />

      <!-- Сообщение если ничего не выбрано -->
      <div v-else class="welcome-message">
        <h2>Добро пожаловать!</h2>
```

```
        <p>Выберите пример для изучения из навигации выше.</p>
      </div>
    </main>

    <footer class="app-footer">
      <p>Vue 3 + Vite • Практика 27</p>
    </footer>
  </div>
</template>

<script>
// Импортируем наши компоненты, пока файлов нет - должно быть закомментировано
/*import ReactiveDemo from './components/ReactiveDemo.vue'
import ConditionalListDemo from './components/ConditionalListDemo.vue'
import EventComputedDemo from './components/EventComputedDemo.vue'
*/

import { ref } from 'vue'

export default {
  name: 'App',

  // Регистрируем компоненты чтобы использовать их в шаблоне
  components: {
//    ReactiveDemo,
//    ConditionalListDemo,
//    EventComputedDemo
  },

  setup() {
    // Текущий активный демо-компонент
    const currentDemo = ref('reactive')

    return {
      currentDemo
    }
  }
}
</script>

<style>
/* Глобальные стили */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f5f5f5;
  color: #333;
  line-height: 1.6;
}
```

```
#app {
  min-height: 100vh;
  display: flex;
  flex-direction: column;
}

/* Стили шапки */
.app-header {
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  color: white;
  padding: 2rem;
  text-align: center;
}

.app-header h1 {
  margin-bottom: 0.5rem;
  font-size: 2.5rem;
}

.app-header p {
  opacity: 0.9;
  font-size: 1.1rem;
}

/* Навигация */
.navigation {
  display: flex;
  justify-content: center;
  gap: 1rem;
  padding: 1.5rem;
  background-color: white;
  box-shadow: 0 2px 10px rgba(0,0,0,0.1);
  flex-wrap: wrap;
}

.nav-button {
  padding: 0.75rem 1.5rem;
  border: 2px solid #667eea;
  background-color: transparent;
  color: #667eea;
  border-radius: 25px;
  cursor: pointer;
  font-size: 1rem;
  transition: all 0.3s ease;
}

.nav-button:hover {
  background-color: #667eea;
  color: white;
  transform: translateY(-2px);
}

.nav-button.active {
```

```
  background-color: #667eea;
  color: white;
}

/* Основное содержимое */
.main-content {
  flex: 1;
  padding: 2rem;
  max-width: 1200px;
  margin: 0 auto;
  width: 100%;
}

.welcome-message {
  text-align: center;
  padding: 4rem 2rem;
  color: #666;
}

.welcome-message h2 {
  margin-bottom: 1rem;
  color: #333;
}

/* Подвал */
.app-footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1rem;
  margin-top: auto;
}
</style>
```

Пока код с подключением компонент закомментирован, но по мере добавления открываем комментарии.

Пример 1: Реактивность и двустороннее связывание

Давайте создадим простой компонент, который демонстрирует основы реактивности Vue. Реактивность - это способность Vue автоматически обновлять интерфейс при изменении данных.

Создайте файл `src/components/ReactiveDemo.vue`:

```
<template>
  <div class="demo-container">
    <h2>Пример 1: Реактивность и v-model</h2>

    <p>Директива v-model создает двустороннее связывание между элементом формы и
данными компонента.</p>
```

```
<!-- Простое текстовое поле с v-model -->
<div class="input-group">
  <label>Введите ваше имя:</label>
  <input
    v-model="userName"
    placeholder="Например: Иван"
    class="text-input"
  >
  <p>Привет, <strong>{{ userName }}</strong>!</p>
</div>

<!-- Textarea с v-model -->
<div class="input-group">
  <label>Опишите ваши интересы:</label>
  <textarea
    v-model="userBio"
    placeholder="Расскажите о себе..."
    class="text-area"
    rows="3"
  ></textarea>
  <p>Длина текста: {{ userBio.length }} символов</p>
</div>

<!-- Select с v-model -->
<div class="input-group">
  <label>Выберите технологию:</label>
  <select v-model="selectedTech" class="select-input">
    <option value="vue">Vue.js</option>
    <option value="react">React</option>
    <option value="angular">Angular</option>
  </select>
  <p>Вы выбрали: {{ getTechName(selectedTech) }}</p>
</div>

<!-- Чекбокс с v-model -->
<div class="input-group">
  <label>
    <input type="checkbox" v-model="isSubscribed">
    Получать уведомления
  </label>
  <p v-if="isSubscribed">Вы подписаны на уведомления ✓</p>
  <p v-else>Вы не подписаны на уведомления</p>
</div>

<!-- Группа радио-кнопок -->
<div class="input-group">
  <label>Уровень опыта:</label>
  <div>
    <label>
      <input type="radio" v-model="experienceLevel" value="beginner">
      Начинающий
    </label>
    <label>
      <input type="radio" v-model="experienceLevel" value="intermediate">
```

```
            Средний
          </label>
          <label>
            <input type="radio" v-model="experienceLevel" value="advanced">
            Продвинутый
          </label>
        </div>
        <p>Ваш уровень: {{ getExperienceText(experienceLevel) }}</p>
      </div>

      <!-- Отладочная информация -->
      <div class="debug-info">
        <h3>Текущее состояние данных:</h3>
        <pre>{{ JSON.stringify({
          userName,
          userBio,
          selectedTech,
          isSubscribed,
          experienceLevel
        }, null, 2) }}</pre>
      </div>
    </div>
  </template>

  <script>
  import { ref } from 'vue'

  export default {
    name: 'ReactiveDemo',

    setup() {
      // ref создает реактивную переменную
      // Когда значение меняется, Vue автоматически обновляет шаблон
      const userName = ref('')
      const userBio = ref('')
      const selectedTech = ref('vue')
      const isSubscribed = ref(false)
      const experienceLevel = ref('beginner')

      // Методы для форматирования данных
      const getTechName = (tech) => {
        const techMap = {
          'vue': 'Vue.js',
          'react': 'React',
          'angular': 'Angular'
        }
        return techMap[tech] || tech
      }

      const getExperienceText = (level) => {
        const levelMap = {
          'beginner': 'Начинающий разработчик',
          'intermediate': 'Разработчик со средним опытом',
          'advanced': 'Опытный разработчик'
```

```
      }
      return levelMap[level] || level
    }

    // Возвращаем все переменные и методы, чтобы они были доступны в шаблоне
    return {
      userName,
      userBio,
      selectedTech,
      isSubscribed,
      experienceLevel,
      getTechName,
      getExperienceText
    }
  }
}
</script>

<style scoped>
.demo-container {
  max-width: 600px;
  margin: 20px auto;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 8px;
  background-color: #f9f9f9;
}

.input-group {
  margin-bottom: 20px;
}

label {
  display: block;
  margin-bottom: 5px;
  font-weight: bold;
}

.text-input, .text-area, .select-input {
  width: 100%;
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 14px;
}

.debug-info {
  margin-top: 30px;
  padding: 15px;
  background-color: #e9ecef;
  border-radius: 4px;
}

pre {
```

```
  background-color: white;
  padding: 10px;
  border-radius: 4px;
  overflow-x: auto;
}
</style>
```

Пример 2: Условный рендеринг и списки

Теперь рассмотрим директивы v-if, v-for и v-show для управления отображением элементов.

Создайте файл `src/components/ConditionalListDemo.vue`:

```
<template>
  <div class="demo-container">
    <h2>Пример 2: Условный рендеринг и списки</h2>

    <!-- Форма для добавления технологий -->
    <div class="add-form">
      <input
        v-model="newTechName"
        placeholder="Введите название технологии"
        class="text-input"
        @keyup.enter="addTechnology"
      >
      <button @click="addTechnology" class="add-button">
        Добавить
      </button>
    </div>

    <!-- Переключение видимости с v-show -->
    <div class="controls">
      <button @click="showCompleted = !showCompleted" class="toggle-button">
        {{ showCompleted ? 'Скрыть' : 'Показать' }} завершенные
      </button>
      <button @click="sortBy = sortBy === 'name' ? 'date' : 'name'" class="toggle-
button">
        Сортировать по: {{ sortBy === 'name' ? 'названию' : 'дате' }}
      </button>
    </div>

    <!-- Условный рендеринг с v-if -->
    <div v-if="technologies.length === 0" class="empty-state">
      <p>Список технологий пуст. Добавьте первую технологию!</p>
    </div>

    <!-- Отображение списка с v-for -->
    <div v-else class="tech-list">
      <div
        v-for="tech in sortedTechnologies"
        :key="tech.id"
```

```
        class="tech-item"
        :class="{ completed: tech.completed }"
      >
        <div class="tech-info">
          <h3>{{ tech.name }}</h3>
          <span class="tech-date">Добавлено: {{ formatDate(tech.createdAt) }}
</span>
        </div>

        <div class="tech-actions">
          <!-- v-show для переключения видимости -->
          <button
            v-show="!tech.completed"
            @click="completeTechnology(tech.id)"
            class="complete-button"
          >
            Завершить
          </button>

          <button
            @click="removeTechnology(tech.id)"
            class="remove-button"
          >
            Удалить
          </button>
        </div>
      </div>
    </div>

    <!-- Статистика -->
    <div class="stats">
      <h3>Статистика:</h3>
      <p>Всего технологий: {{ technologies.length }}</p>
      <p>Активных: {{ activeCount }}</p>
      <p>Завершенных: {{ completedCount }}</p>

      <!-- Условный рендеринг с v-if/v-else -->
      <div v-if="completedCount > 0" class="progress-section">
        <p>Прогресс: {{ progressPercentage }}%</p>
        <div class="progress-bar">
          <div
            class="progress-fill"
            :style="{ width: progressPercentage + '%' }"
          ></div>
        </div>
      </div>
      <div v-else>
        <p>Начните изучать технологии чтобы увидеть прогресс!</p>
      </div>
    </div>
  </div>
</template>

<script>
```

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

export default {
  name: 'ConditionalListDemo',

  setup() {
    // Реактивные данные
    const newTechName = ref('')
    const technologies = ref([])
    const showCompleted = ref(true)
    const sortBy = ref('date')

    // Вычисляемые свойства
    const activeCount = computed(() =>
      technologies.value.filter(tech => !tech.completed).length
    )

    const completedCount = computed(() =>
      technologies.value.filter(tech => tech.completed).length
    )

    const progressPercentage = computed(() => {
      if (technologies.value.length === 0) return 0
      return Math.round((completedCount.value / technologies.value.length) * 100)
    })

    const sortedTechnologies = computed(() => {
      const techsToShow = showCompleted.value
        ? technologies.value
        : technologies.value.filter(tech => !tech.completed)

      return [...techsToShow].sort((a, b) => {
        if (sortBy.value === 'name') {
          return a.name.localeCompare(b.name)
        } else {
          return new Date(b.createdAt) - new Date(a.createdAt)
        }
      })
    })

    // Методы
    const addTechnology = () => {
      if (!newTechName.value.trim()) return

      technologies.value.push({
        id: Date.now(),
        name: newTechName.value.trim(),
        completed: false,
        createdAt: new Date().toISOString()
      })

      newTechName.value = ''
    }
```

```
    const removeTechnology = (id) => {
      technologies.value = technologies.value.filter(tech => tech.id !== id)
    }

    const completeTechnology = (id) => {
      const tech = technologies.value.find(tech => tech.id === id)
      if (tech) {
        tech.completed = true
      }
    }

    const formatDate = (dateString) => {
      return new Date(dateString).toLocaleDateString('ru-RU')
    }

    return {
      newTechName,
      technologies,
      showCompleted,
      sortBy,
      activeCount,
      completedCount,
      progressPercentage,
      sortedTechnologies,
      addTechnology,
      removeTechnology,
      completeTechnology,
      formatDate
    }
  }
}
</script>

<style scoped>
.demo-container {
  max-width: 600px;
  margin: 20px auto;
  padding: 20px;
}

.add-form {
  display: flex;
  gap: 10px;
  margin-bottom: 20px;
}

.text-input {
  flex: 1;
  padding: 8px;
  border: 1px solid #ccc;
  border-radius: 4px;
}

.add-button, .toggle-button {
```

```
  padding: 8px 16px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.controls {
  display: flex;
  gap: 10px;
  margin-bottom: 20px;
}

.tech-list {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.tech-item {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 15px;
  border: 1px solid #ddd;
  border-radius: 8px;
  background-color: white;
}

.tech-item.completed {
  background-color: #d4edda;
  border-color: #c3e6cb;
}

.tech-info h3 {
  margin: 0 0 5px 0;
}

.tech-date {
  font-size: 12px;
  color: #666;
}

.tech-actions {
  display: flex;
  gap: 5px;
}

.complete-button {
  padding: 5px 10px;
  background-color: #28a745;
  color: white;
  border: none;
```

```
  border-radius: 4px;
  cursor: pointer;
}

.remove-button {
  padding: 5px 10px;
  background-color: #dc3545;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.empty-state {
  text-align: center;
  padding: 40px;
  color: #666;
}

.stats {
  margin-top: 30px;
  padding: 20px;
  background-color: #f8f9fa;
  border-radius: 8px;
}

.progress-bar {
  width: 100%;
  height: 20px;
  background-color: #e9ecef;
  border-radius: 10px;
  overflow: hidden;
}

.progress-fill {
  height: 100%;
  background-color: #28a745;
  transition: width 0.3s ease;
}
</style>
```

Пример 3: Работа с событиями и вычисляемыми свойствами

В третьем примере рассмотрим обработку событий и мощные вычисляемые свойства.

Создайте файл `src/components/EventComputedDemo.vue`:

```
<template>
  <div class="demo-container">
    <h2>Пример 3: События и вычисляемые свойства</h2>
```

```
    <!-- Таймер изучения -->
    <div class="timer-section">
      <h3>Таймер изучения</h3>
      <div class="timer-display">
        {{ formatTime(elapsedTime) }}
      </div>
      <div class="timer-controls">
        <button @click="startTimer" :disabled="isRunning" class="timer-button
start">
          Старт
        </button>
        <button @click="pauseTimer" :disabled="!isRunning" class="timer-button
pause">
          Пауза
        </button>
        <button @click="resetTimer" class="timer-button reset">
          Сброс
        </button>
      </div>
    </div>

    <!-- Список сессий изучения -->
    <div class="sessions-section">
      <h3>Сессии изучения</h3>

      <div v-if="studySessions.length === 0" class="no-sessions">
        <p>Сессий пока нет. Запустите таймер!</p>
      </div>

      <div v-else class="sessions-list">
        <div
          v-for="session in sortedSessions"
          :key="session.id"
          class="session-item"
        >
          <div class="session-info">
            <span class="session-date">{{ formatDate(session.date) }}</span>
            <span class="session-duration">{{ formatTime(session.duration) }}
</span>
          </div>
          <button @click="removeSession(session.id)" class="delete-session">
            ×
          </button>
        </div>
      </div>

      <!-- Статистика сессий -->
      <div class="sessions-stats">
        <p>Всего сессий: {{ totalSessions }}</p>
        <p>Общее время: {{ formatTime(totalStudyTime) }}</p>
        <p>Средняя продолжительность: {{ formatTime(averageSessionTime) }}</p>
      </div>
    </div>
```

```
    <!-- Быстрые действия с модификаторами событий -->
    <div class="quick-actions">
      <h3>Быстрые действия</h3>
      <div class="action-buttons">
        <!-- Модификатор .prevent предотвращает действие по умолчанию -->
        <button @click.prevent="addQuickSession(30)" class="action-button">
          +30 мин
        </button>
        <button @click.prevent="addQuickSession(60)" class="action-button">
          +1 час
        </button>
        <!-- Модификатор .once срабатывает только один раз -->
        <button @click.once="addOneTimeSession" class="action-button special">
          Одноразовая сессия
        </button>
      </div>
    </div>
  </div>
</template>

<script>
import { ref, computed, onUnmounted } from 'vue'

export default {
  name: 'EventComputedDemo',

  setup() {
    // Таймер
    const elapsedTime = ref(0)
    const isRunning = ref(false)
    let timerInterval = null

    // Сессии изучения
    const studySessions = ref([])

    // Вычисляемые свойства для статистики
    const totalSessions = computed(() => studySessions.value.length)

    const totalStudyTime = computed(() =>
      studySessions.value.reduce((total, session) => total + session.duration, 0)
    )

    const averageSessionTime = computed(() => {
      if (totalSessions.value === 0) return 0
      return Math.round(totalStudyTime.value / totalSessions.value)
    })

    const sortedSessions = computed(() =>
      [...studySessions.value].sort((a, b) => new Date(b.date) - new Date(a.date))
    )

    // Методы таймера
    const startTimer = () => {
      isRunning.value = true
```

```
      timerInterval = setInterval(() => {
        elapsedTime.value += 1
      }, 1000)
    }

    const pauseTimer = () => {
      isRunning.value = false
      if (timerInterval) {
        clearInterval(timerInterval)
        timerInterval = null
      }
    }

    const resetTimer = () => {
      pauseTimer()

      // Сохраняем сессию если прошло больше 30 секунд
      if (elapsedTime.value >= 30) {
        studySessions.value.push({
          id: Date.now(),
          date: new Date().toISOString(),
          duration: elapsedTime.value
        })
      }

      elapsedTime.value = 0
    }

    // Методы для сессий
    const removeSession = (sessionId) => {
      studySessions.value = studySessions.value.filter(session => session.id !==
sessionId)
    }

    const addQuickSession = (minutes) => {
      const duration = minutes * 60 // переводим в секунды
      studySessions.value.push({
        id: Date.now(),
        date: new Date().toISOString(),
        duration: duration
      })
    }

    const addOneTimeSession = () => {
      addQuickSession(45)
      alert('Одноразовая сессия добавлена! Эта кнопка больше не сработает.')
    }

    // Вспомогательные методы
    const formatTime = (seconds) => {
      const mins = Math.floor(seconds / 60)
      const secs = seconds % 60
      return `${mins.toString().padStart(2, '0')}:${secs.toString().padStart(2,
'0')}`
```

```
    }

    const formatDate = (dateString) => {
      return new Date(dateString).toLocaleString('ru-RU')
    }

    // Очистка при размонтировании компонента
    onUnmounted(() => {
      if (timerInterval) {
        clearInterval(timerInterval)
      }
    })

    return {
      elapsedTime,
      isRunning,
      studySessions,
      totalSessions,
      totalStudyTime,
      averageSessionTime,
      sortedSessions,
      startTimer,
      pauseTimer,
      resetTimer,
      removeSession,
      addQuickSession,
      addOneTimeSession,
      formatTime,
      formatDate
    }
  }
}
</script>

<style scoped>
.demo-container {
  max-width: 600px;
  margin: 20px auto;
  padding: 20px;
}

.timer-section {
  text-align: center;
  margin-bottom: 30px;
  padding: 20px;
  background-color: #f8f9fa;
  border-radius: 10px;
}

.timer-display {
  font-size: 3rem;
  font-weight: bold;
  margin: 20px 0;
  color: #007bff;
```

```
}

.timer-controls {
  display: flex;
  gap: 10px;
  justify-content: center;
}

.timer-button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-weight: bold;
}

.timer-button:disabled {
  opacity: 0.5;
  cursor: not-allowed;
}

.timer-button.start {
  background-color: #28a745;
  color: white;
}

.timer-button.pause {
  background-color: #ffc107;
  color: black;
}

.timer-button.reset {
  background-color: #dc3545;
  color: white;
}

.sessions-section {
  margin-bottom: 30px;
}

.sessions-list {
  display: flex;
  flex-direction: column;
  gap: 10px;
  margin-bottom: 20px;
}

.session-item {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px 15px;
  background-color: white;
  border: 1px solid #ddd;
```

```
  border-radius: 5px;
}

.session-info {
  display: flex;
  flex-direction: column;
}

.session-date {
  font-size: 14px;
  color: #666;
}

.session-duration {
  font-weight: bold;
  color: #007bff;
}

.delete-session {
  background: none;
  border: none;
  font-size: 20px;
  cursor: pointer;
  color: #dc3545;
}

.no-sessions {
  text-align: center;
  padding: 20px;
  color: #666;
  font-style: italic;
}

.sessions-stats {
  padding: 15px;
  background-color: #e9ecef;
  border-radius: 5px;
}

.quick-actions {
  text-align: center;
}

.action-buttons {
  display: flex;
  gap: 10px;
  justify-content: center;
  flex-wrap: wrap;
}

.action-button {
  padding: 10px 20px;
  background-color: #6c757d;
  color: white;
```

```
    border: none;
    border-radius: 5px;
    cursor: pointer;
  }

  .action-button.special {
    background-color: #17a2b8;
  }
  </style>
```

Контрольная работа №5

Задание для контрольной работы №5 разбито на несколько составляющих: 4 балла ставится за выполнения задания по практикам 27-28 4 балла ставится дополнительно за выполнение каждой из 4 контрольных работ (по 1 баллу за каждую работу).

Задание для практик 27-28: "Генератор цветовых палитр"

Общая концепция

Разработайте интерактивное веб-приложение на Vue.js для создания, управления и экспорта цветовых палитр. Приложение должно стать полезным инструментом для дизайнеров, разработчиков и всех, кто работает с визуальным контентом.

Общее описание проекта

Создайте генератор цветовых палитр, который позволяет пользователям:

Генерировать гармоничные цветовые схемы

Настраивать и редактировать палитры

Анализировать цвета на соответствие стандартам доступности

Сохранять и организовывать коллекции палитр

Экспортировать цвета в различные форматы для использования в проектах

Проект реализуется в течение двух практических занятий с постепенным наращиванием функциональности.

Практика 27: Базовый функционал

Цель: Создать работающий прототип генератора с основными возможностями.

Обязательные функции:

1. Система генерации палитр

Кнопка "Случайная палитра" - генерирует 5 гармоничных цветов

Отображение палитры в виде горизонтальной полосы цветовых карточек

Для каждого цвета показывать HEX-значение (например, #FF6B6B)

2. Управление отдельными цветами

Клик по цветовой карточке копирует HEX-значение в буфер обмена

Индикация успешного копирования (всплывающее уведомление)

Возможность "закрепить" понравившийся цвет при генерации новой палитры

3. Базовые инструменты настройки

Выбор количества цветов в палитре (3, 5, 7)

Переключение между форматами отображения (HEX, RGB)

Локальное сохранение текущей палитры в localStorage

4. Простой просмотрщик

Превью палитры в mockup интерфейса (кнопка, карточка, заголовок)

Переключение светлого/тёмного фона для тестирования контраста

Технические требования:

- Использование Vue 3 Composition API
- Реактивное управление состоянием цветов
- Применение директив v-for, v-if, v-model
- Вычисляемые свойства для преобразования цветов
- Обработка событий кликов и ввода

Критерии успеха практики 27:

- ☑ Приложение генерирует случайные палитры
- ☑ Цвета отображаются с HEX-значениями
- ☑ Работает копирование в буфер обмена
- ☑ Палитра сохраняется между перезагрузками
- ☑ Интерфейс интуитивно понятен

Практика 28: Продвинутый функционал (Advanced Features)

Цель: Превратить прототип в полнофункциональный инструмент.

Дополнительные функции:

1. Продвинутая генерация

Генерация на основе базового цвета (пользователь выбирает основной цвет)

Различные типы палитр: аналогичная, монохромная, триада, комплементарная

Генерация по "настроению": спокойные, энергичные, профессиональные палитры

2. Инструменты анализа и доступности

Проверка контрастности между цветами по стандарту WCAG

Показ уровня доступности (AA, AAA, недостаточно)

Подбор акцентных цветов для выбранной палитры

Графическое представление цветового круга

3. Управление библиотекой палитр

Сохранение палитр в именованные коллекции

Поиск и фильтрация по названиям и тегам

Возможность редактирования сохранённых палитр

Создание избранных коллекций

4. Экспорт и интеграция

Экспорт в форматы: CSS variables, SCSS variables, Tailwind config

Генерация готового CSS кода с цветами

Превью палитры в различных UI-компонентах

Создание шаринговых ссылок на палитры

Технические требования:

- Использование Vue Router для навигации
- Компонентный подход с передачей props
- Работа с v-model в кастомных компонентах
- Использование watchers для реактивных изменений
- Интеграция с внешними API для цветовых преобразований

Критерии успеха практики 28:

☑ Реализована генерация по типу палитры

☑ Работает проверка контрастности и доступности

☑ Пользователь может сохранять и организовывать палитры

☑ Доступен экспорт в несколько форматов

- [x] Приложение имеет полноценную навигацию
