# Лабораторная работа №1

## Цель работы

Познакомиться с основным функционалом фреймворка Vue.js

## Введение

Vue.js - прогрессивный фреймворк для создания пользовательских интерфейсов. Vue является масштабируемым фреймворком и поддерживает, как постепенное внедрение, так и создание полноценных SPA (одностраничный приложений). С помощью высокоуровневых фреймворков, например Nuxt.js, вы также можете создать SSR(Отрисовка на стороне сервера) приложение. Ключевое отличие между SPA и SSR в том, что в SPA весь js код, включая код, формирующий страницы выполняется на стороне браузера, а в SSR формирование страницы происходит на стороне сервера. Это ускоряет загрузку страницы, а также улучшает индексацию вашего сайта. Но в этой лабораторной работе  мы поговорим об SPA.

Создание любого приложения Vue начинается с создания нового экземпляра приложения с помощью функции `createApp`.
Экземпляр приложения используется для регистрации «глобальных» вещей, которые будут затем использоваться компонентами внутри этого приложения.

Опции, передаваемые в `createApp`, используются для настройки корневого компонента. При монтировании приложения он используется как стартовая точка для отрисовки.

Приложению требуется примонтироваться в DOM-элемент. Для этого используется метод `mount`, в который передается селектор корневого DOM-элемента.

Для хранения данных в компоненте используется метод `data`.

Все свойства, объявленные в `data`, доступны через экземпляр компонента, а также поддерживают реактивность. Реактивность - свойство, означающее, что Vue следит за изменениями значений переменных и перерисовывает нужные участки DOM в соответствии с изменениями.

В компоненте свойство `data` должно быть функцией. Vue вызывает эту функцию на этапе создания нового экземпляра компонента. Она должна вернуть объект, который затем Vue обернёт в свою систему реактивности и сохранит в экземпляре компонента как `$data`.

Для добавления методов в экземпляр компонента используется опция `methods`. Значением должен быть объект, который будет содержать все необходимые методы.

Vue автоматически привязывает значение `this` к методам таким образом, чтобы оно указывало на экземпляр компонента. Это гарантирует, что в методе всегда сохраняется правильное значение `this`, даже при использовании в качестве обработчика события или коллбэка. Следует избегать использования стрелочных функций при определении `methods`, так как это не позволит Vue привязать корректное значение `this`

Как и все остальные свойства экземпляра компонента, `methods` доступны в шаблоне компонента

## Ход работы.

В этой лабораторной работе мы реализуем фотоальбом нашего сайта с помощью Vue.js
Для начала подключим Vue к нашей странице. Мы будем подключать Vue 3 с помощью CDN. В html файл добавим.

```
<script src="https://unpkg.com/vue@next"></script>
```

Вы также можете скачать данный js файл, разместить его у себя в папке проекта и добавить в index.html.

После подключения этого файла нам будет доступен объект Vue в наших файлах js, подключенных к этом же HTML файлу.

В примере, который мы рассмотрим приложение Vue будет реализовывать только блок с фотографиями.
Для того чтобы создать веб-приложение Vue нам необходимо создать объект, содержащий параметры нашего приложения: компоненты, шаблоны, переменные, и передать его в метод createApp объекта Vue, который создает экземпляр приложения Vue и привязать его к корневому элементу с помощью метода mount. В метод mount мы передаем селектор корневого элемента в index.html.

Создадим корневой элемент приложения в HTML файле:

```
<div id="app"></div>
```

В javascript файле:

```
const PhotoAlbum = {}
const app = Vue.createApp(PhotoAlbum)
app.mount('#app')
```

Для того чтобы выводить фотографии в фотоальбом, нам потребуется массив. Для примера мы создадим массив объектов, хранящих информацию о каждой фотографии. В данной лабораторной работе мы рассмотрим подход к описанию компонентов - Options API. При этом подходе, объект, содержащий конфигурацию приложения, разбивается на поля и методы с семантическими названиями. Например, поле methods будет содержать все методы нашего приложения или компонента, поле computed будет содержать все вычисляемые свойства, в методе data нам необходимо вернуть объект, содержащий все данные нашего компонента или приложения. В данном случае мы возвращаем массив объектов, содержащих информацию о каждой фотографии.

```
data() {
   return {
       photos: [
           {
               title: 'Фотография с космонавтом',
               photo: '../assets/img/image1.jpg',
               alt: 'Альтернативный текст',
               comment: 'Комментарий к фотографии'
           }, {
               title: 'Фотография с пирамидами',
               photo: '../assets/img/image2.jpg',
               alt: 'Альтернативный текст',
               comment: 'Комментарий к фотографии'
           },
...
       ]
   }
}
```

Переменные, объявленные в возвращаемом объекте функцией data - реактивны.
Фотографии будем выводить в div, помещенный в корневой элемент.

```
<div id="app">
```

```
    <div>
</div>
</div>
```

Так как Vue приветствует компонентный подход к созданию приложений, создадим компонент фотографии.
Для создания компонента, нам необходимо передать название и объект конфигурации компонента в метод component объекта нашего приложения. Метод component позволяет нам регистрировать глобальные компоненты нашего приложения, для их дальнейшего применения. Передаем в него два параметра: название нашего компонента, и объект, содержащий конфигурацию компонента, на подобии с созданием самого приложения, описанного выше.

app.component('album-item', {})

Наш компонент элемента фотоальбома требует входной параметр, который будет содержать информацию об изображении. Во vue входные параметры описываются в поле props.
Назовем наш параметр package.
props: ['package'],
Далее нам необходимо описать верстку нашего компонента. Это делается в поле template

```
template:
    `
    <li>
        <img :src="package.photo" :alt="package.alt">
    </li>
`
```

: перед атрибутами тэга img - сокращенная форма v-bind директивы, которая позволяет задавать значения тэгов динамически (<img v-bind:src="package.photo" v-bind:alt="package.alt">).
В данном случае мы установили атрибут src равным полю photo входного параметра data.photo, а атрибут alt - полю package.alt.
Теперь, когда у нас есть компонент элемента фотоальбома, мы можем вывести наши фотографии. Для этого познакомимся с еще одной директивой Vue - v-for. v-for позволяет нам итерировать объекты, массивы и даже числа из javascript.

Применим эту директиву на наш элемент фотоальбома в ul в корневом элементе.

```
<album-item
       v-for="(photo,i) in photos"
       :key="i"
       :package="photo"
>
</album-item>
```

Vue требует уникальный ключ для каждого итерируемого элемента. В данном случае мы на каждой итерации присваиваем key значение i - номера итерации.
Не забываем про входной параметр. Входной параметр инициализируется как атрибут.
Передаем в package значение photo.

Итак, на данный момент мы имеем элементарный вывод картинок в браузере. Давайте расширим функционал. Добавим возможность открывать картинки в отдельном окне, и переключать их по нажатию на соответствующие кнопки.

Создадим компонент полноэкранной картинки. Как и в прошлый раз нам необходимо вызвать метод component,

```
app.component('img-popup', {})
```

Данный компонент будет принимать 2 входных параметра. Массив с фотографиями и индекс открытой фотографии.

```
props: ['photos', 'index'],
```

Шаблон будет выглядеть так:

```
template:
   `
   <teleport to="body">
   <div class="img_popup" @click.self="$emit('close')">
```

```
        <button type="button" class="to_left"
@click="previos">&#8249;</button>
        <div class="content">
            <img :src="photos[id].photo" :alt="photos.alt">
            <div class="text">
                <h2>{{photos[id].title}}</h2>
                <p>{{photos[id].comment}}</p>
            </div>
        </div>
        <button type="button" class="to_right"
@click="next">&#8250;</button>
    </div>
</teleport>
`,
```

Здесь мы знакомимся с новым встроенным компонентом Vue - teleport. teleport позволяет сделать родительским элементом, помещенных внутрь него элементов, элемент, указанный в атрибуте to. В данном случае мы “телепортируем” наш попап в body.
@click - это сокращенная запись от директивы v-on:click. v-on: директива, для перехвата событий. Vue поддерживает все события из html, а также позволяет создавать пользовательские с помощью метода $emit()
.self - один из модификаторов перехвата события. При использовании данного модификатора событие будет перехвачено, только при клике именно на этот элемент.

При нажатии на img_popup мы генерируем событие close, которое позже обработаем в родительском компоненте.

При нажатии на кнопки мы вызываем методы previous и next для переключения фотографий назад и вперед соответственно.
Методы в компонентах Vue описываются в поле methods.

```
methods: {
    previous: function () {
```

```
        if (!this.id) {
            this.id = this.$props.data.length - 1
        } else {
            this.id--
        }
    },
    next: function () {
        if (this.id === this.$props.data.length - 1) {
            this.id = 0
        } else {
            this.id++
        }
    }
}
```

Vue приветствует принцип одностороннего потока (нельзя напрямую изменять значения входных параметров), нам необходимо добавить атрибут id внутри объекта, возвращаемого в методе data(), для того чтобы взаимодействовать с ним.

```
data() {
    return {
        id: this.$props.index
    }
},
```

Добавим значение index в компоненте приложения

```
data() {
    return {
```

```
        index: -1,
        photos: [
            {
                title: 'Фотография с космонавтом',
                photo: '../assets/img/image1.jpg',
                alt: 'Альтернативный текст',
                comment: 'Комментарий к фотографии'
            }, {
                title: 'Фотография с пирамидами',
                photo: '../assets/img/image2.jpg',
                alt: 'Альтернативный текст',
                comment: 'Комментарий к фотографии'
            },
...
        ]
    }
}
```

Применим созданный компонент в корневом элементе.

```
<div id="app">
    <div class="album">
        <album-item
                v-for="(photo,i) in photos"
                :key="i"
                :data="photo"
        ></album-item>
    </div>
    <img-popup v-if="index !== -1" :data="photos" :index="index"
@close="index=-1"></img-popup>
</div>
```

v-if директива условной отрисовки. Если значение index будет отлично от -1 то компонент будет добавлен в DOM дерево. В параметр data передаем массив photos, в index - значение переменной index. Здесь же мы обрабатываем событие ‘close’, которое генерировали внутри компонента img-popup. При перехвате данного события значение index будет устанавливаться -1, что будет скрывать popup.

В компонент элемента фотоальбома нам необходимо добавить обработчик нажатия, который будет генерировать событие нажатия на картинку.

```
<span class="album-item" @click="$emit('click')">
```

```
   <img :src="data.photo" :alt="data.alt">
</span>
```

Обработаем это событие в приложении.

```
<album-item
       v-for="(photo,i) in photos"
       :key="i"
       :data="photo"
       @click="index = i"
></album-item>
```

При срабатывании обработчика мы устанавливаем index значение i - индекса картинки.

Готовый проект https://codepen.io/ghosterbeef/pen/JjJapYm

## Контрольные вопросы

Как создать приложение Vue 3 и связать его с DOM?
Как хранятся данные в компонентах Vue?
Перечислите встроенные директивы Vue 3.
Для чего нужен компонент teleport?
В чем отличие computed свойства от метода из methods и наблюдателя watch?
Что такое Односторонний поток данных?