## Содержание

## Тренировочное задание

**Цель:** ознакомится с основным функционалом фреймворка Vue.js.

## Краткие теоретические сведения

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.Регистры общего назначения (РОН), кроме аккумулятора могут объеди- няться в пары (B-C,D-E и H-L) и использоваться как 16-битовые регистры.

## Порядок выполнения лабораторной 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, разметить 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 этому 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 – реактивные. Фотографии будем выводить в div, помещенные 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 парметр 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 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 – индекс картинки.

## Контрольные вопросы

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

## Лабораторная работа №6

**Цель:** изучить принцип работы с формами и их методы валидации.

## Краткие теоретические сведения

Во Vue существует понятие – *двусторонняя привязка*. Проще всего объяснить двустороннюю привязку с помощью полей ввода.

Допустим у нас есть текстовое поле - <input type=”text” /> У данного поля есть атрибут value

Если мы используем директиву v-bind:value=”” то мы привяжем значение поля к модели, однако при изменении данного значения в поле, модель изменяться не будет. Чтобы избежать этой проблемы существует двусторонняя привязка. Чтобы ее реализовать необходимо использовать директиву v-model=””

Например.

Добавим в data поле textValue: “”

Добавим в html текстовое поле.

```
<input type=”text” v-model=”textValue”/>
```

Добавим в html заголовок h2, в который будем выводить значение поля ввода.

```
<h2>Значение: {{ textValue }}</h2>
```

При изменении значения текстового поля будет изменяться значение заголовка.

### Модификаторы событий

Вы уже знаете, что во Vue.js можно устанавливать обработчики событий с помощью директивы v-on и ее краткой записи @. Также Vue предоставляет удобные модификаторы событий для упрощения работы с последними.

Для применения модификаторов их необходимо через точку указать после названия обрабатываемого события. Например:

```
@click.stop.prevent=”i++”
```

Данная запись является короткой формой для:

```
v-on:click=”(e) => {
     e.preventDefault() e.stopPropagation()
     i++
}”
```

•.*stop* – всплытие события click будет остановлено;

•.*prevent* – поведение по умолчанию не будет выполняться;

•.*capture* – событие, нацеленное на внутренний элемент, обрабатывается до обработки элементом , на котором находится обработчик;

•.*self* – вызов обработчика только в случае наступления события непосредственно на данном элементе (то есть не на дочернем компоненте);

•.*once* – обработчик будет вызван максимум 1 раз;

•.*passive* – сообщает браузеру, что для события не будет предотвращаться поведение по умолчанию.

## Ход работы

Для начала необходимо создать проект с помощью интерфейса Vue CLI и убрать ненужные компоненты и материалы, созданные автоматически.

Затем в App.vue создадим модель формы с полями: Имя, Фамилия, Возраст, Пол, Фреймворки.

```
data: () => ({
     formData: {
```

```
            name: “”,
            surname: “”,
            age: null,
            sex: “”,
            frameworks:[],
      }
})
```

Добавим варианты для пола и фреймворков.

```
data: () => ({
      ...
      sexOptions: Object.freeze({
            M: “M”
            F: “F”
      }),
      frameWorksOptions: Object.freeze({
            VUE: “Vue”,
            ANGULAR: “ANGULAR”,
            SVELTE: “SVELTE”,
            REACT: “REACT”
      })
})
```

Создадим верстку для данной модели.

```
<form>
      <input type=”text” v-model=”formData.name”/>
      <input type=”text” v-model=”formData.surname”/>
```

```
    <input type=”number” v-model=”formData.age”/>

    <input type=”radio” v-model=”formData.sex” v-for=”option in sexOptions” :value=”option” />

    <input type=”checkbox” v-model=”formData.frameworks” v-for=”option in frameWorksOptions” :value=”option”/>

</form>
```

Создадим валидацию:

```
computed: {

    errors() {

        const errors = [];

        if (!this.formData.name)

            errors.push({ field: "name", message: "Введите имя!" });

        else errors.filter((error) => error.field !== "name");

        if (!this.formData.surname)

            errors.push({ field: "surname", message: "Введите Фамилию!" });

        else errors.filter((error) => error.field !== "surname");

        if (!this.formData.age && typeof this.formData.age !== "number")

            errors.push({ field: "age", message: "Укажите возраст!" });

        else errors.filter((error) => error.field !== "age");

        if (!this.formData.name)

            errors.push({ field: "sex", message: "Укажите пол!" });

        else errors.filter((error) => error.field !== "sex");

        return errors;

    },
```

```
}
```

И будет блокировать отправку до тех пор, пока есть ошибки. Ошибки можно выводить снизу формы.

## Порядок выполнения работы

Разработать дизайн, создать веб-приложение, содержащее форму с указанными в ходе работы полями. Реализовать валидацию формы. При нажатии на кнопку отправить и корректно заполненной форме выводить сообщение.

`

## Контрольные вопросы

1. Как реализовать двустороннее связывание?
2. Как отменить поведение по умолчанию у формы?
3. Перечислите модификаторы v-model.

## Лабораторная работа №7

**Цель:** изучить принципы маршрутизации и управления состоянием приложения.

### Краткие теоретические сведения

Для реализации роутинга в веб-приложении в экосистеме Vue существует плагин vue-router, который включает все необходимые функции и возможности.

Этот плагин необходимо устанавливать отдельно или включить его в пресет во время создания проекта.

Чтобы создать экземпляр vue-router-а необходимо создать файл, например, router.js, в который необходимо импортировать методы: createRouter – создает экземпляр роутера и метод, создающий необходимую историю маршрутов. Рекомендуется использовать Web History. Этот метод использует привычную всем маршрутизацию с помощью слеша.

Затем необходимо создать массив маршрутов.

Массив маршрутов — это массив объектов со следующим минимальным набором полей: path – путь маршрута, component – компонент, который необходимо отрисовать при переходе по данному маршруту.

Создаем экземпляр роутера:

```
const router = createRouter({
     history: createWebHistory(process.env.BASE_URL),
     routes
})
```

и экспортируем его

```
export default router
```

Чтобы роутер заработал его необходимо зарегистрировать в качестве плагина через экземпляр приложения.

Перед монтированием экземпляра приложения в DOM – дерево необходимо добавить

```
import router from ‘./router.js’
app.use(router)
```

После этих манипуляций в каждом компоненте станут доступны 2 встроенных компонента: router-view и router-link, а также 2 поля объекта компонента $router и $route.

router-view – компонент, вместо которого будут рендериться компонент текущего маршрута.

router-link – компонент, который является ссылкой для vue-router. $router – указатель на экземпляр роутера.

$route – указатель на экземпляр маршрута.

Для навигации по маршрутам необходимо указывать параметр to в компоненте router-link.

Существует альтернативный вариант осуществления навигации. Существуют методы $router.push и $router.replace, которые могут изменять маршруты в зависимости от переданных параметров. Это полезно если вам необходимо осуществлять навигацию из модели, а не из представления.

router-link после компиляции становится тегом a, к которому применяются различные классы. Например, активная ссылка получает класс router- linkexact-active.

**Защитник маршрута.**

Во vue-router есть защитник маршрута beforeEach который вызывается при каждом переходе по ссылке. Данный метод принимает параметром функцию обратного вызова с параметрами: to – маршрут, на который происходит переход, from – маршрут, с которого происходит переход и next – метод, который позволяет осуществить переход по маршруту.

В данном методе, можно, например проверять авторизацию пользователя и перенаправлять его на страницу авторизации. Или вы можете изменить название вкладки браузера, в зависимости от страницы, на которую происходит переход. Для этого необходимо добавить метаинформацию в массив с маршрутами.

```
{
     path: ...,
     meta: {
          title: “Домашняя страница”
     }
}
```

И в функции обратного вызова присваивать document.title значение to.meta.title.

**Управление состоянием приложения.**

В экосистеме Vue существует библиотека для управления состоянием под названием vuex. Как и vue-router она не включена изначально в Vue, однако вы можете добавить ее вручную или добавить при создании проекта.

**Зачем нужен vuex?**

При создании крупного приложения, разработчик может столкнуться с проблемой глубокой вложенности компонентов или наоборот множеством компонентов на одном уровне вложенности, которые используют одни и те же данные. В такой ситуации и приходит на помощь vuex. Vuex позволяет вынести хранилище общих данных из компонентов, во внешний общедоступный объект, и при этом позволяет реализовать механизмы работы с ними: реактивные геттеры, мутации и т.д.

**Применение vuex.**

Чтобы создать хранилище создадим файл store.js, в который импортируем метод createStore из vuex.

Данный метод принимает параметром объект конфигурации хранилища, а возвращает экземпляр хранилища, который мы экспортируем.

Чтобы использовать хранилище в приложении, необходимо его зарегистрировать подобно роутеру.

```
import store from ‘./store.js’
```

```
app.use(store)
```

После регистрации хранилища, в каждом компоненте станет доступно свойство $store – указатель на экземпляр хранилища.

**Конфигурация vuex.**

Объект конфигурации хранилища содержит следующие поля:

- *state* – объект с реактивными данными.
- *getters* – объект, который содержит все реактивные геттеры данных.
- *mutations* – объект, который содержит все методы синхронного взаимодействия с данными.
- *actions* – объект, который содержит асинхронные методы взаимодействия с данными с помощью мутаций.
- *modules* – объект, который содержит модули хранилища.
- *state* – очень похож на data в компонентах, кроме того, что взаимодействовать с ним необходимо через интерфейсы, а не напрямую для поддержания реактивности.

Существует вспомогательный метод mapState, который создает вычисляемые свойства из выбранных полей хранилища в компоненте.

getters – аналог вычисляемых свойств у компонентов. Методы, которые возвращают какие-либо вычисления и преобразования хранилища без его изменения.

Существует вспомогательный метод mapGetters, который создает вычисляемые свойства из выбранных геттеров.

mutations – это методы, которые вносят изменения в данные хранилища. Мутации не могут быть асинхронными. Чтобы вызвать мутацию необходимо вызвать метод commit.

```
this.$store.commit('имяМутации', payload)
```

Существует вспомогательный метод mapMutations, который создает методы из выбранных мутаций.

actions – это методы, которые могут быть асинхронными, а также могут

изменять хранилище с помощью мутаций. Чтобы вызвать действие необходимо вызвать метод dispatch

```
this.$store.dispatch('имяДействия', payload)
```

Существует вспомогательный метод mapActions, который создает методы из выбранных действий.

modules – модули полезны, когда ваше хранилище становится огромным. Модули помогают разделить хранилище на зоны ответственности и улучшают читабельность кода.

## Порядок выполнения работы

Создать приложение, которое будет содержать несколько страниц с произвольным наполнением, навигацию с помощью vue-router, а также продемонстрируйте работу vuex в вашем приложении.

## Контрольные вопросы

1. Что такое vuex?
2. Что такое vue-router?
3. Чо такое navigation guard?
4. Зачем нужно хранилище состояния?
5. Что такое мутации? В чем их особенность?
6. Какие вспомогательные методы vuex вы знаете?