Advanced Web Programming

Lab4 - Introduce to Vue 3

The purpose of the lab is to learn the fundaments concepts of Vue 3. You will learn how to:

- Create a simple Vue application
- Render lists
- Render DOM elements conditionally
- Handle events
- Use attribute binding
- Use computed properties
- Use methods

The lab is divided into 2 sections. In the first section you practice main concepts doing simple exercises with my help. In the second you do assignments yourself based on the assignments specification.

Section 1.

The aim of the exercises in that section is to practice the Vue3.

In the catalog Vue3_startTemplate you have the starting set of files for all exercises. You have index.html, style.css and app.js. App.js file is now empty. This is the file where you write Vue code in the lab that will manipulate your application.

File index.html:

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>EFRAI lab4 - Vue3 Start Template</title>
  <link      href="https://fonts.googleapis.com/css2?family=Jost:wght@400;700&display=swap"
   rel="stylesheet"    />
  <link rel="stylesheet" href="styles.css" />
  <script src="https://unpkg.com/vue@next" defer></script>
  <script src="app.js" defer></script>
 </head>
 <body>
     <div id="app">
```

```
    </div>
  </body>
</html>
```

There are three important lines in the above code:

- <script src="https://unpkg.com/vue@next"></script> - with this line, you import the Vue library from a CDN. Using the CDN link is the easiest way to import Vue into your application and fiddle with it.
- <div id="app"></div> - this div represents the application. You mount the application into the DOM by using this div with the class of app.
- <script src="./app.js"></script> - with this line, you import the app.js file, which stores your Vue code. This is the file where you write Vue code that will manipulate your application.

1. Create the first Vue app

Clone the contents of the directory Vue3_startTemplate to the directory excercise1 .

The next step is to go to app.js and create a Vue application. Write the following code in your file:

```
const app = Vue.createApp({
    data() {
        return {
           greeting: 'Hello, Welcome in the first Vue app'
       }
   }
})
app.mount('#app');
```

In the above code, Vue.createApp creates a new instance of Vue. While creating the Vue instance, you also pass an “Options” object that allows you to configure the application. For example, you added a greeting field in your Vue app, which you can access in the HTML file.
However, the newly created application needs to be mounted into a DOM element. If you don’t mount the application into DOM, you will not be able to use Vue. Try removing the app.mount('#app') line and then try to access the greeting.

Now that you created a Vue application, you can access the greeting property. Let’s change the index.html file to display greeting Value. Write the following code in the #app div:

```
<div id="app">
        <h1>{{ greeting }}</h1>
</div>
```

You can access proprieties from your Vue app by using double curly braces. The double curly braces will be replaced with the value of the `greeting` property.

Test the application. You just created your first Vue application, so congratulations to you

Exercise 1. Display in Excercise1 app your name and your age too.

Exercise 2: Create in index.html the second div section. Try to display using interpolation your name in that div. What is a result? What are the conclusions?

Exercise 3. Check what happen when you write next expression in div id=”app” section.

{{2 +2}}

{{ name.length}}

{{ a = 2+3}}

{{ windows.location.href}}

2. Attribute Binding

Extend our application about new data items. We add new item to data section. Let item name is imgURL.

```
const app = Vue.createApp({
    data() {
        return {
            .........
            imgURL: "https://ss7.vzw.com/is/image/VerizonWireless/iPhone8-
Svr?$png8alpha256$&hei=520",
            imgDescription: 'An image with a mobile’,
            .........
        }
    },
    .........
})
```

I replaced the rest of the code with *”…”* for readability purposes. The code is the same as earlier, but with the additions of:

- imgURL
- imgDescription

Now, going further, there is another addition to the HTML:

```
<img :src="imgURL" :alt="imgDescription" width="500" height="350">
```

As you can see, we use the image URL and description specified in the Vue instance. You can create this reactive bond between the attribute and arguments thanks to the v-bind directive. The `v-bind` directive allows developers to bind an attribute dynamically to an expression. It enables us to use dynamic values rather than hard-coded values. Now, you might be confused because there is no `v-bind` in the code. That's because there is a shorthand for `v-bind` , which is simply the colon symbol - :.

```
<img :src="imgURL" :alt="imgDescription" width="500" height="350">
```

can be re-written using `v-bind` as follows:

```
<img v-bind:src="imgURL" v-bind:alt="imgDescription" width="500" height="350">
```

Thanks to this reactivity system of Vue, you can update the image URL and description in your Vue instance, and the HTML will update too automatically.

Exercise 4. Add tag a to our exercise1 app. Add url to the data object. Use v-bind to bind the url to anchor tag's href attribute. Test app, check link .

As you can imagine, since there's so many different HTML attributes, there are many use cases for v-bind. For example, you might be binding a description to an alt attribute or binding a URL to an href as we do in above code. We can binding some dynamic styles to a class or style attribute, disabling and enabling a button, and so on.

3. Event Handling

Create new project Exercises2 by cloning file from catalog GR_events-starting-code. As you see in template we have in app two button: one to increment, the second decrement counter.

In Vue, you can use the v-on directive to listen to DOM events and run a piece of code when they are triggered. The v-on directive has a shorthand value as well, which is the symbol "@".

```
<button @click="logic to run">Do something</button>
```

When someone clicks the button, Vue triggers the method you specify - doSomething in this case. Of course, you can replace the method with code directly. For instance, you could increment a variable - @click="counter = counter + 1", but we want to use method.

Exercise 5. Implement method add and reduce in Exercise2 app. Display in app current value of counter.

A Vue method is a function associated with the Vue instance. Methods are defined inside the methods property:

```
new Vue({
    methods: {
      handleClick: function() {
              alert('test')
      }
    }
})
```

In Vue methods are implement in new field. You have a methods field on the Vue instance. This field allows you to create methods and use them in your application.

```
const app = Vue.createApp({
  data() {
    return {
      counter: 0,
    };
  },
  methods: {
    add() {
      this.counter = this.counter + 1;
    },
    reduce() {
      this.counter = this.counter - 1;
      // this.counter--;
    }
  }
});
app.mount('#events');
```

index.html code:

```
<h2>Events in Action</h2>
      <button v-on:click="add">Add</button>
      <button v-on:click="reduce">Reduce</button>
      <p>Result: {{ counter }}</p>
```

Event modifiers

Instead of messing with DOM "things" in your methods, tell Vue to handle things for you:

- @click.prevent call event.preventDefault()
- @click.stop call event.stopPropagation()

- @click.passive makes use of the passive option of addEventListener
- @click.capture uses event capturing instead of event bubbling
- @click.self make sure the click event was not bubbled from a child event, but directly
- happened on that element
- @click.once the event will only be triggered exactly once

All those options can be combined by appending on modifier after the other.

Exercise 6. Display in exercises2 a starting value (initial) of counter.

Exercise 7. Extend exercises2 by adding input of type text to add your name. Display your name with your surname bellow row with input field. Let your name and surname will be displayed only when you press key enter.

Exercise 8. In exercises2 add next input of type text and submit button. When you click submit let app display alert message with value from input value. Let value of counter, name and surname and new input value not to be clear after that. My suggestion: use prevent Event modifiers.

At the beginning in point nr 2, we learned about v-bind, which creates a one-way binding, from the data to the template. When working with forms, however, this one way binding isn't enough. We also need to be binding from the template to the data.

For example, when a user inputs their name into an input field, we want to record and store that value in our data. The v-model directive helps us achieve this, creating two-way data binding.

Exercise 9. In exercises2 test v-model binding by using new text input, reset button and property in data section, Value of property display in app by interpolation. Let reset button reset value of property.

Vue allows you to loop over an array using the v-for directive. The v-for directive has the form of tag in tags. It's always easier to understand with an example. Look at the following code:

```
const app = Vue.createApp({
   data() {
        return {
            tags: ['Vue', 'Front-end', 'JavaScript']
        }
    }
})
```

```
<ul>
    <li v-for="tag in tags">
        {{ tag }}
    </li>
</ul>
```

The above code loops over the “tags” array and renders each tag on the page. tags represents the array from the Vue application, whereas the tag is an individual element of the array.
This is what you see when you run the application:

Unique key attribute
Whenever you loop over arrays in Vue, you should give each DOM element a unique key. Usually, the key of each element is the element’s id.
By providing the key attribute, you allow Vue to keep track of nodes’ identity as things updated in the application. As a result, Vue can re-use and re-order existing elements. Besides that, it also improves the performance of the application. You can modify the code to include the index field as well. The “index” field represents the position of the element in the array.

```
<ul>
    <li v-for="(tag, index) in tags" :key="index">
        {{ tag }}
    </li>
</ul>
```

However, if you have an array of objects, such as:

```
tags:
[
    { id: 1, name: 'Vue' },
    { id: 2, name: 'Front-end' },
    { id: 3, name: 'JavaScript' },
 ]
```

You can provide the id of the object as the key. The loop would look as follows:

```
<ul>
    <li v-for="tag in tags" :key="tag.id">
       {{ tag }}
    </li>
</ul>
```

:key ->:key is the shorthand of v-bind:key, and it’s used to bind the item’s id to the “key” attribute. That is, each item has a unique “key”, which you specify in the :key field. In this example, each tag has its id as the key.

Conditional rendering
Sometimes, you want to display elements on a page based on a condition. That is, use if statements to decide whether to render an element or not. Or to render it based on a condition. One example would be - show all the programming courses if the rating is over 4 stars. To do so, you can use the v-if directive, which renders a block only when the expression returns a truthy value

```
const app = Vue.createApp({
    data() {
        return {
            available: true,
            tags: [
                { id: 1, name: 'Vue' },
                { id: 2, name: 'Front-end' },
                { id: 3, name: 'JavaScript' },
            ]
        }
    }
})
```

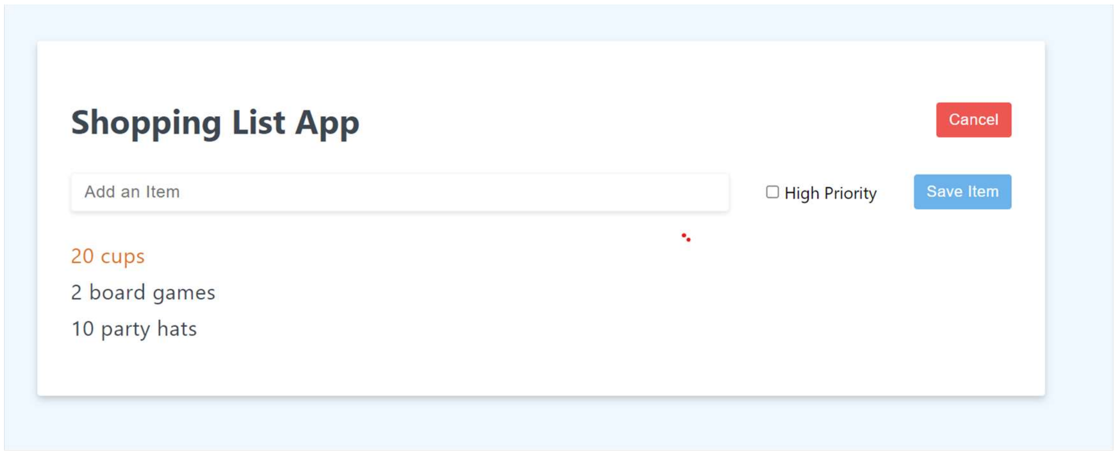
Note the new field called “available”. The “available” field is set to true. However, you can keep changing it between “true” and “false” to see how the conditional rendering works.

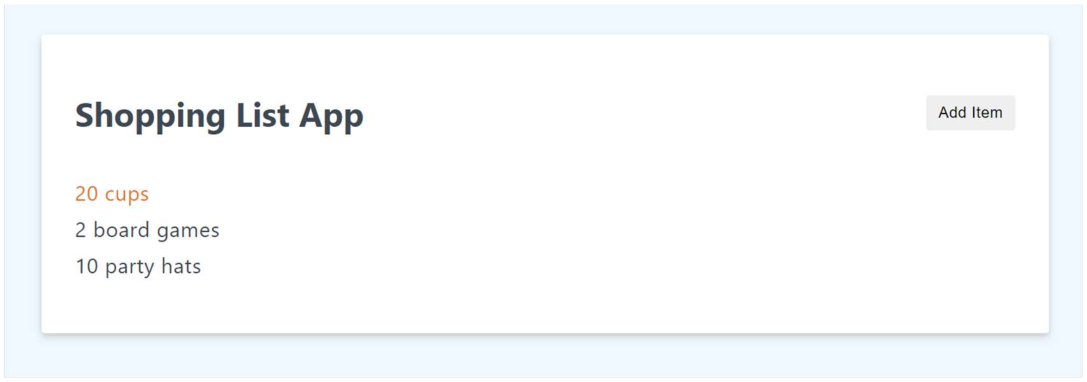
```
<p v-if="available">You can buy the course!</p>
<p v-else>The course is not available to buy!</p>
```

The code above displays You can buy the course! if the available is true. Otherwise, it displays The course is not available to buy!.
v-if takes the field you want to evaluate. If the field evaluates to a truthy value, it displays the first option. Otherwise, it shows the second options. As an exercise, change the available field to false and see what happens!

Section 2.

Exercises 10. Create shopping list app. Goal app is to manage shopping list. You can add new items to shopping list or disable this possibility (example of screen from example app you can seen below). Button save item add to list data from text input. Checkbox hight priority set level od priority added item. When checkbox is seleted items is added in red color (as a very important position in the list). When you click button cancel possibility of add items to list is off. Then you can see only Add item button. When you click this button your app display first screen and enable to add items to list.





Homework.

Create shop application. In application you have a collection at least 8 elements. Every element has name, number of items, unit price and url of product photo.

Display list of products. In one row we have photo, name, price and number of product items. There should be 2 buttons + and - next to each product, allowing you to add a product to the basket list or return it. When you add item of product to basket number of product is decrement. If you return item from basket to list number of product increment.

Basket display the value of the products ordered.

If the value of the product in the basket is 0, a different message should be displayed than when the number of available products is greater than 0.

If the amount of the product drops to zero, the - button should be hidden. After all, we do not want to reserve a product that we no longer have.

When the amount of the product is approaching 0 (e.g. from 3 downwards), it should be marked in a graphic way, e.g. a different background, font color, font size or other visual way.

Similarly, a distinction should be made between the product with the lowest unit price and the highest - with an additional border covering a given product - green - the most expensive, red - the cheapest.

Also display the total number of currently available products - if it is more than 10, it should be displayed on a green background, if below 10 - on a red background.