Unit 5 - Vue.js - Node.js

Classwork: The classwork for this unit should be saved in a new folder:
your_repo/unit_5/vuejs_nodejs/
Homework: Homework should be saved here: **your_repo/homework/unit_5/**

*** Day 01 ***

What is a component in Vue.js?

Components in Vue.js are like modules in Python. You will be able to create a component, which will be a piece of your HTML code and you will be able to reuse it multiple times (for example for a list of data), and even if you use it only once, it will be useful to separate the code to keep it clean.

But for that, we will need to deploy our **Vue.js app** with **Node.js,** which will **compile our Vue.js app in JavaScript.**

Let's install **npm**, a package manager for Node.js and is included as a recommended feature when you install the Node.js. We are going to use Python to install npm because there is a python package to help us create a npm virtual env like we do for python.

Install npm

1. Create a python virtual environment
2. Activate the virtual env
3. `pip install nodeenv` (this package will help to create a nodejs virtual environment, like we have with python)
4. Install nodejs 20.11.1 (the latest LTS(long-term) version at this time), anywhere you want.
   nodeenv --node=20.11.1 --prebuilt env_node_20.11.1
5. Deactivate your python => deactivate
6. Activate your node env
   source env_node_20.11.1/bin/activate
7. Done!

Install Vue.js

Go here for info: https://vuejs.org/guide/quick-start.html

We are going to create a Vue app via script. You need to run this command on the folder you are going to have your project.

```
npm create vue@latest
```

Select No for everything except the last 2 options: ESLint and Prettier

Go to the new folder created by Vue.js and type:

```
npm install
```

Npm is like the pip in Python.

Vue.js (via the script npm create vue@latest), has created a **package.json**. Here is where you set up the js packages you want to use. This is similar to the requirement file we use in Python.

In package.json, we have the Vue.js package as dependency. We also have some devDependencies, we have some packages here to check the syntax and formatting. There is a package named vite. **Vite will be used to create a front-end server.** Any modification you will do, it is going to be refreshed immediately, like the live server we used previously.

Npm is going to install the packages in the **node_modules** folder. You can check, there are a lot of packages there.

One advice in case you use google drive (or OneDrive for microsoft): do not install npm package on this type of drive because it is going to take a long time to sync (there are a lot, a lot of small files). Like the python environment, too. But in the python environment you can install it anywhere you want, activate it, and install packages as you want. With nodejs, you do not have this flexibility, so be aware. Nodejs is going to install a node_modules folder in the same directory as your js project. So annoying. I searched on how to have this node_modules outside of the projects, but did not find anything.

Once npm has installed the packages, we can **run the vite server**, by running:

****
npm run dev
****

It is going to tell you in which port it is running. Mine is running at http://localhost:5173/

In the package.json, there is a **scripts section,** and we see that the **dev command is calling vite**. Vite is going to look at the vite.config.js and will use the file index.html as an entry point for the html. I am not very familiar with vite yet. By the way, vite is a French word that means fast.

Vue js has been setting up things for us.

The index.html page is calling to the src/main.js
In the src folder, you have your vue js app.
In the main.js you have the vuejs app and it is mounted.

Rename the folder src, as src_vuejs, and let's create a new folder src.

Copy the main.js file from src_vuejs to src. Remove the line about the CSS.

Create an App.vue in the folder src.
And let's copy the hello-world example (https://vuejs.org/examples/#hello-world):

```
<!--
Say Hello World with Vue!
-->

<script setup>
import { ref } from 'vue'

// A "ref" is a reactive data source that stores a value.
// Technically, we don't need to wrap the string with ref()
// in order to display it, but we will see in the next
// example why it is needed if we ever intend to change
// the value.
const message = ref('Hello World!')
</script>

<template>
  <h1>{{ message }}</h1>
</template>
```

Save it, you should see the new page refreshed automatically on your page served by the vite server. (http://localhost:5173/ for me)

I let you discover the other examples: https://vuejs.org/examples/

*** Day 02 ***

Components

So now, we can create a complete Vue.js application, we are not limited in the functionalities anymore.

Components are very useful in vue js because it is like an object you can reuse. You may need to implement communication with a component, and here is when things begin to get messy. So let's start.

Let's rewrite the App.vue:

```
<template>
  <h1>{{ message }}</h1>
  <h2>{{ test() }}</h2>
</template>

<script>
import { ref } from 'vue'

// A "ref" is a reactive data source that stores a value.
// Technically, we don't need to wrap the string with ref()
// in order to display it, but we will see in the next
// example why it is needed if we ever intend to change
// the value.
export default {
  name: "App",
  components: {
  },
  data() {
      return {
      message: ref('Hello World!')
      }
  },
  methods: {
      test(){
            return "This is a test"
      }
  },
  computed: {
  },
}
</script>
```

For the component, it is better to name it with two words, so you are sure it does not conflict with a name used by the HTML language.

Let's create a folder component inside the src folder.
Let's create a file MovieItem.vue and insert a blank structure for the component

```
<template>
</template>

<script>
export default {
      components: {
      },
      data() {
            return {
            }
      },
      methods: {
      },
      computed: {
      },
}
</script>
```

Let's add a static template, just to try.

```
<template>
      <h2>This is a movie item</h2>
</template>
```

Now we are going to import this component into our App.vue.

Let's go back to App.vue.
In the script tag, let's import our component

```
<script>
import { ref } from 'vue'
import MovieItem from './components/MovieItem.vue'
. . .
```

Let's add our component inside the key components in our app:

```
export default {
  name: "App",
  components: {
     MovieItem,
  },
. . .
```

So far, so good. Now let's use it into the template of our app to see if it works:

```
<template>
  <h1>{{ message }}</h1>
  <h2>{{ test() }}</h2>
  <movie-item></movie-item>
</template>
```

We are going to use it as an HTML element. And we will be able to pass arguments also.
We can add another movie-item element.

```
<template>
  <h1>{{ message }}</h1>
  <h2>{{ test() }}</h2>
  <movie-item></movie-item>
  <movie-item></movie-item>
</template>
```

Let's check our webpage HTML code (right click and inspect).

We see some h2 elements instead of our movie-item: this is because vue-js is taking care of everything, and on the html code you see the template of movie-item.

For the moment, you could say that the structure of the component is the same as our vue-js app. So everything you learn, you can apply it for components.

****

Exercise 1:

On our component MovieItem.vue, every time I left-click on "This is a movie item", I would like to have a counter right below it, showing how many times I clicked on it. Then, when the component is loaded, we initialize at 0 and we do not show it on our webpage

What should be added to the <h2> tag?
What element you may use to show the counter?
Do you need a data variable? A method?

****

And you see when you click on one component, it does not interact with the other one. Each component is independent from others. We defined the data inside the component, there is still no communication with the exterior, so this component is living its life apart.

You can add styling in the .vue files.

Open the App.vue, and at the end, you can put some style between <style></style>

```
<style>
  h1 {
      color: blue
  }

  h2 {
      color: purple
  }
</style>
```

At this moment the component is going to inherit from the style of the parent vue app.
Also you can defined also some style on the component:

```
<style>
  h1 {
      color: red
  }

  h2 {
      color: green
  }
</style>
```

But you will see that the parent app is overriding this styling. (the parent app styles have priority). When you look at the html code on the browser (inspect), in the head, you will see the css style from the app is coming after the one from the component.

If you want to define some specific styling for the component, it is better to do it with a class anyway, like you would do in a normal html page.

Communication between components and with the main Vue application

Now, besides showing the components on the webpage, let's make them share the data we want between each other. Here we would like to give each component some data for a movie (Name, year, description…)

One way to communicate data from a parent to a child component is called **props** (short name of properties - html element custom attribute).

We have the 2 movie-item tag components in our Vue application, we are going to use the props attribute to define some data along the movie-item tag for example (just an example, maybe we are not going to do exactly like this, maybe we are going to pass an object, but it is just to show the concept):

```
<movie-item name="Matrix" :year="1999" :actors='["Keanu Reeves",
"Carrie-Anne Moss"]'></movie-item>
```

What is the difference between : and no :. The same explanation as we may have seen before, with :, we bind it with a javascript expression, and with no :, you can pass a static value as a string.

You can put that in the code, vue js is not going to complain. Vue js is going to pass these "arguments" to the component, but the component is not expecting anything, so it moves on.

We are going to introduce the **props properties on the components object.**

Let's go to the MovieItem.vue

Below components (the order does not matter, it is just I like to keep the components first, to see what components the current component is using), create a props key, and list the name of the props inside of it, and also we can specify the type of the object we are expecting. You do not have to specify the type of object, at this moment, just define props as a list, and name the props as a list of strings.

```
export default {
      components: {
      },
      props: {
            name: String,
            year: Number,
            actors: Array,
            },
      ,.....
```

****

Exercise 2:

Considering these props as if they were defined as in data, display the information in the template component. It is ok to replace the code you currently have in the template section in MovieItem. Keep the add counter functionality.

****

*** Day 03 ***

****

Exercise 3:

Let's create a data array variable with movies information, in App.vue

```
 data() {
        return {
                movies_list: [
                        {
                                'name': 'Matrix',
                                'year': 1999,
                                'actors': ['Keanu Reeves', 'Carrie-Anne Moss',],
                        },
                        {
                                'name': 'Amelie Poulain',
                                'year': 2001,
                                'actors': ['Audrey Tautou', 'Mathieu Kassovitz'],
                        },
                        {
                                'name': 'Les visiteurs',
                                'year': 1993,
                                'actors': ['Jean Reno', 'Christian Clavier',],
                        }
                ],
        message: ref('Hello World!')
        }
 },
```

In the App.vue, remove the <movie-item></movie-item> we manually set it up and make it now work with this list.

Hint: You may use v-for in a template tag inside the main template.

```
<template>
  <h1>{{ message }}</h1>
  <h2>{{ test() }}</h2>
 <template v-for="movie in movies_list">
       ……
 </template>
</template>
```

Or you may use a v-for directly:

```
<movie-item v-for="movie in movies_list" …… ></movie-item>
```

By the way, in the MovieItem, if you have the v-for on the <ul>, please change it should be in the <li> tag:

```
<ul>
      <li v-for="actor in actors">{{actor}}</li>
</ul>
```

****

It is also a good idea to use :key in the v-for "To give Vue a hint so that it can track each node's identity, and thus reuse and reorder existing elements, you need to provide a unique key attribute for each item" (https://vuejs.org/guide/essentials/list)
Let's add a key to our data.

```
{'id': 1,
 'name': 'Matrix',
 'year': 1999,
 'actors': ['Keanu Reeves', 'Carrie-Anne Moss',],
  'like': true,},
….

<movie-item v-for="movie in movies_list" :key="movie['id']"
   :id="movie['id']"
   :name="movie['name']" :year="movie['year']" :actors="movie['actors']" ></movie-item>
```

So now we know how to create components, and give information to a component. And you see it is easy to reuse components. That is great.

Change a prop in the component

Let's add a prop for this purpose 'like' (MovieItem)

```
props: {
        id: Number,
        name: String,
        year: Number,
        actors: Array,
        like: Boolean,
        },
```

Let's put 'like' in the data array as false for each item,

{'id': 1,
'name': 'Matrix',
'year': 1999,
'actors': ['Keanu Reeves', 'Carrie-Anne Moss',],
'like': false,},
…..

Let's add the 'like' when we call movie-item from the template.

```
<movie-item v-for="movie in movies_list" :key="movie['id']"
  :id="movie['id']"
  :name="movie['name']" :year="movie['year']" :actors="movie['actors']"
  :like="movie['like']"></movie-item>
```

So far so good.

In the MovieItem.vue template, let's add a condition to see when 'like' is true:

```
<h2 v-on:click="addCounter">{{ name }} ({{year}})<template v-if="like"> (like)</template></h2>
```

****

Exercise 4:

Create a button and an event to change the value 'like'.
If 'like' is false, the button should show "like", if it is true, "remove like"
Do it using a method changeLike() { }

When checking if it works, look at the console. Is there any error?

****

Important: Vuejs is not going to let you change a prop from the child component. (https://vuejs.org/guide/components/props.html#one-way-data-flow). Be careful when you change an object, because sometimes vuejs will let you do that, but have other consequences. Read the web page for more info.

If you want to change the value inside the component, without the need to update the data on the main app, then you can do the following:

You can create a new variable in data and assign that value (MovieItem).

```
data() {
      return {
            clicked_counter: 0,
            movieLike: this.like
      }
},
```

And then replace 'like' by movieLike in your method changeLike() and in the template. It will work. But in this case you did not replace the value in the data source.

Communication from the child to the parent

We are going to use component events.

Let's change the value of the 'like' in the data source in App.vue (add some true values).

Let's go back to the MovieItem.vue, replace your code inChangeLike() with the following:

```
changeLike() {
            this.$emit('change-like', this.id)
      }
```

You are going to **emit an event to the parent component**. Set the name of the event and also you can set multiple arguments on this event.

Also, set this event name in the emits properties:

```
export default {
   components: {
   },
   emits: ['change-like'],
. . . .
```

Let's go to the App.vue and set up the event in the movie-item tag:

```
<movie-item v-for="movie in movies_list" :key="movie['id']"
  :id="movie['id']"
  :name="movie['name']" :year="movie['year']" :actors="movie['actors']"
  :like="movie['like']"
   @change-like="changeLike"></movie-item>
```

The @change-like comes from the event name you set up on the child component.
And "changeLike" is going to be the method you define in the parent component. So, your App.vue must look like this:

And now let set up the method (App.vue):

```
methods: {
    test(){
      return "This is a test"
    },
    changeLike(movieId){
      console.log('test movieId', movieId)
      const movie = this.movies_list.find(movie => movie.id ==
movieId)
      movie.like = !movie.like
      console.log(movie)
    }
  },
```

So far this is not working, so what do we need now to make this work?

If you install the vuejs plugin on firefox (or chrome if you are using chrome, but I advise you to use firefox because otherwise you are giving more power to google and it has enough power right now, so we do not need to give it more.)

After installing the vuejs plugin, reload the html page, right click inspect to have the developer bar, and you see you have a new menu vuejs. Click on it, you can see the data for the .vue page that your html is using.

Click on the App vue. You can see the data stored in the movie_list array. Click on the button of the first movie-item, you see the like did not change, just click on the refresh button of the Vue tab (on the top right, there is a refresh button), and then you see the 'like' value changed.

Q: Why did the component not change from true to false (or from false to true)?

Hint: Does movieLik work exactly as the prop 'like'?
After fixing the previous issue, it should be working properly. Now we know how to create a component, use the component with initial data, and emit events from the child component to the parent component.

If you want to make a prop required, you can. By requiring a prop you are making sure that if other components change the values those props have a value.

```
props: {
       id: {type: Number,
           require: true},
       name: {type: String,
              require: true},
       year: {type: Number,
              require: true},
       actors: {type: Array,
                require: true},
       like: {type: Boolean,
              require: false,
              default: false},
   },
```

Also, you can be more specific and add more arguments. You can tell how many arguments this function is going to emit, and have some console warning for some conditions you can specify, for example:

If you remove the id argument from our emit event:

```
changeLike() {
           this.$emit('change-like')
       }
```

You see a bunch of warnings and errors on the console.

*** Day 04 ***

****

Exercise 5:

Let's create a new component newMovie.vue

Add a form in the template, with inputs: name, year and actors. Ids will be added with the max id you have in the list (+1), and 'like' will be false by default (no need to add it in the form, it is false).

- Add a button to "Add Actor"
- Buttons to "Remove Actor"
- Add a button to "Submit Movie"

After adding a movie, your movie list must be refreshed.

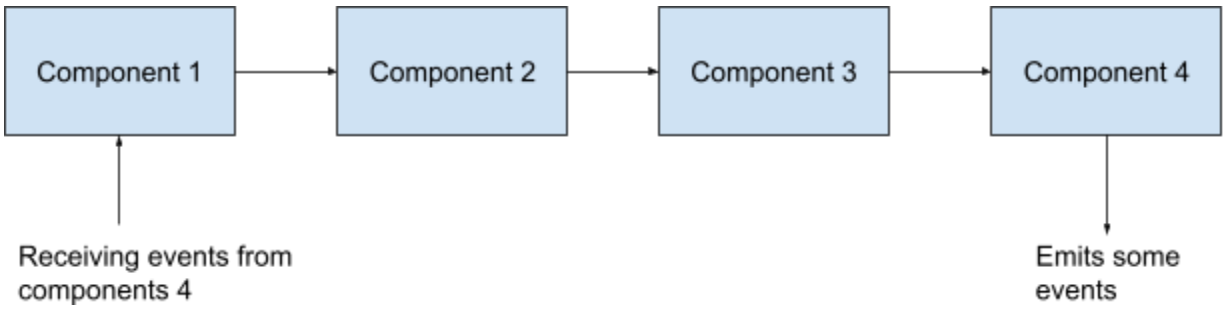
Hint: Use prevent in your form, so your web page will not be refreshed while you work with the form: **`<form @submit.prevent>`**

****

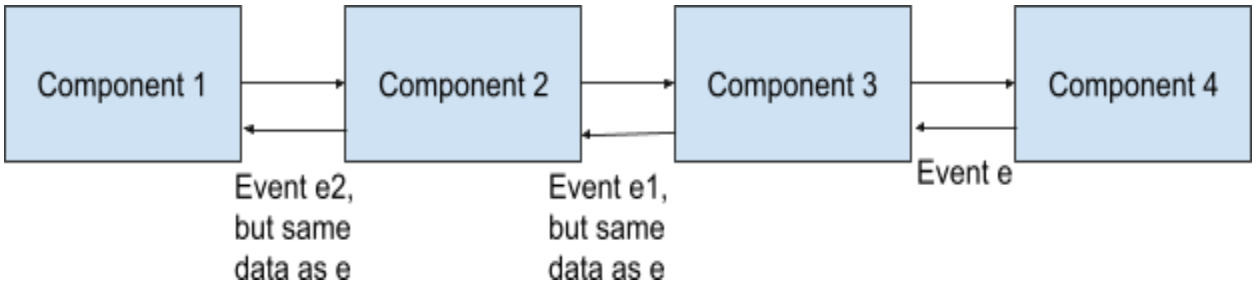
*** Day 05 ***

We have made some progress on vue. We have seen components, which are very useful as it is going to give us the opportunity to reuse some JS code with a lot of useful tools. Doing the same on vanilla JS would be hard, not impossible, but hard, you will need a lot of time. That is why frameworks are so useful, because it is creating a structure/system you can use. Of course, frameworks have some limitations, but you can go over these limitations by expanding it yourself if you want it.

Now let's see a case you can confront: You have component 1, that is going to use component 2, that is going to use component 3, that is going to use component 4. Component 4 will emit an event, but it is component 1 that needs to receive it. Most of the time because component 1 has the main data, and needs to be modified there first for then passed on to the other components.



So, how would you do that?



So you would do something like this. Component 4 fires the event, component 3 picks up the event, but then fires the same data in another event, same thing for component 2, and then component 1 receives the event finally. Components 2 and 3 do not process the event, but are passing along the event; so, component 1 can receive it, because in the chain of communication between component 4 and component 1, it should pass by component 2 and 3.

Provide and Inject

The previous solution would work. However, it is annoying doing that, plus you can make some mistakes or forget something.

Vuejs created a feature for this case. We have 2 new keys, called **provide and inject.**

This feature would provide the data for children components, and then you will be able to inject it on the component you would like (with a parent component that provided the data). No more passing through prop for components. **Important**, you only use provide/inject for data when the components in the middle do not need this data.

Let's see a simple example to show you the concept behind provide/inject. Let's create two new components:
MovieGrid: Will use the movie list
MovieBase: Will be used just to load MovieGrid

MovieBase.vue

```
<template>
	<movie-grid></movie-grid>
</template>
<script>
	import MovieGrid from './MovieGrid.vue'

	export default {
		components: {
			MovieGrid
		},
		props: {

		},
		data() {
			return {
			}
		},
		methods: {
		},
		computed: {
		},
	}
</script>

<style>
</style>
```

In the App.vue, import MovieBase, add a <movie-base> tag in the template and let's add provide, and we will load inside the provide this.movie_list.

```
provide() {
      return {
            movies_list: this.movies_list
      }
  },
```

And in MovieGrid, we are going to inject it:

```
...
props: {

},
inject: ['movies_list'],
data() {
    return {
    }
},
. . .
```

Let's move some code into MovieGrid.vue.

Move the `<movie-item>.....</movie-item>` from the template in App.vue to the template in MovieGrid.app

Do not forget to import MovieItem in MovieGrid.vue

At this point, we should see the movie list on the page. Our provide/inject worked.

For the events we are going to use the same technique, but instead of providing/injecting data, we are going to provide/inject a method.

****

Exercise 6:

In the component MovieItem, when we click anywhere on the template, we are going to show the title of this movie, on the top of the page instead of "This is a test".

Think about how you would do it using provide and inject with a method this time.

****