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.

****