

TP3 - VueJs

Pei LIU

Exercise 1: Create a new Vue project (called vue-oauth-microsoft-graph). Opt for the Vue3 recipe that relies on webpack and babel for the build chain.

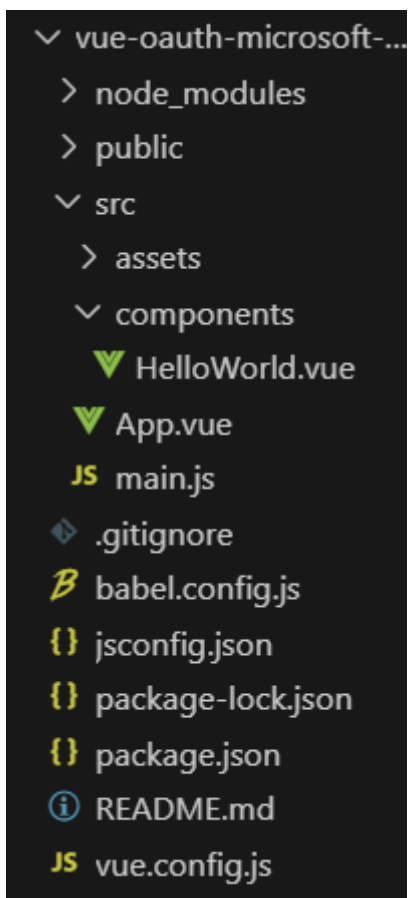
- Install the vue cli globally

```
npm install -g @vue/cli
```

- Create a new project

```
npm create vue-oauth-microsoft-graph
```

The overall of structure for a Vue project



```

  vue-oauth-microsoft-...
  ├── node_modules
  ├── public
  ├── src
  │   ├── assets
  │   ├── components
  │   │   ├── HelloWorld.vue
  │   │   └── App.vue
  │   ├── main.js
  │   ├── .gitignore
  │   ├── babel.config.js
  │   ├── jsconfig.json
  │   ├── package-lock.json
  │   ├── package.json
  │   ├── README.md
  │   └── vue.config.js

```

Question 2: Webpack is internally used by the Vue CLI. Why is it required to deal with both multiple JavaScript files and special extensions like `.vue`?

Vue.js uses both `.js` files and `.vue` single-file components (SFCs) for distinct purposes that complement each other in building modular and maintainable applications.

- **Multiple `.js` files:** Vue.js applications often involve logic spread across multiple JavaScript files to achieve modularity. For example, you may separate routing logic, Vuex store management, utility functions, and API requests into different `.js` files. This organization ensures that logic is reusable, testable, and easier to maintain.
- `.vue` files encapsulate the **template**, **script**, and **style** in a single file. These components enable a more structured approach to component-based development.

Template: Defines the HTML structure of the component.

Script: Handles the business logic, events, and state management within the component.

Style: Scopes the CSS specific to that component to avoid conflicts with global styles.

Question 3: What is the role of babel and how browserslist may configure its output?

Babel in the context of Vue 3 refers to the use of Babel as a **JavaScript compiler** that can transform modern JavaScript (ES6+ or newer syntax) into a version that is compatible with older browsers. In Vue 3 projects, Babel can be integrated to handle this transpilation process, allowing developers to write

modern JavaScript while ensuring that their code works across different environments.

Browserslist determines which browsers to target by parsing a configuration file or a `browserslist` key in `package.json`. Based on the configuration, tools like Babel and Autoprefixer will adapt their transformations to ensure compatibility with the specified browsers. For example, Babel will transpile modern JavaScript features for unsupported browsers, and Autoprefixer will add necessary vendor prefixes for CSS.

Question 4: What is eslint and which set of rules are currently applied?
The eslint configuration may be defined in a `eslint.config.js` or in `package.json` depending on the setup.

ESLint is a static code analysis tool used to find and fix problems in JavaScript code. It helps developers maintain consistent coding styles, avoid bugs, and enforce best practices by running checks against your code based on predefined or custom rules. ESLint can be configured to enforce specific coding standards and catch potential errors, such as syntax issues, variable usage, or code quality concerns.

Error Prevention: Rules that catch syntax errors, improper use of variables, or bad code patterns (e.g., `no-unused-vars`, `no-undef`).

Stylistic Issues: Rules that enforce code style, such as spacing, indentation, and semicolons (e.g., `indent`, `quotes`, `semi`).

Best Practices: Rules that help developers write better code, like enforcing the use of strict equality (`eqeqeq`), or preventing the use of `console.log` in production (`no-console`).

Exercise 2: Run `npm run serve` and open the app in your browser. Remember that npm looks at the `package.json` file (specially the `scripts` object) to find which command to execute.

```
PS D:\Efrei\S7\Interface\TP3_VueJs\vue-oauth-microsoft-graph> npm run serve
```

```
> vue-oauth-microsoft-graph@0.1.0 serve
> vue-cli-service serve
```

```
INFO Starting development server...
```

App running at:

- Local: <http://localhost:8081/>
- Network: <http://10.3.207.11:8081/>

Note that the development build is not optimized.
To create a production build, run `npm run build`.

```
"scripts": {
  "serve": "vue-cli-service serve",
  "build": "vue-cli-service build",
  "lint": "vue-cli-service lint"
},
```

Exercise 3: The newly generated project contains a few placeholders. Clean up your project so it does not contain neither useless assets, nor the hello world. In other words, delete HelloWorld.vue, its related assets and all its references. As at the end of each exercise, the vue cli should not report any error or warning.



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

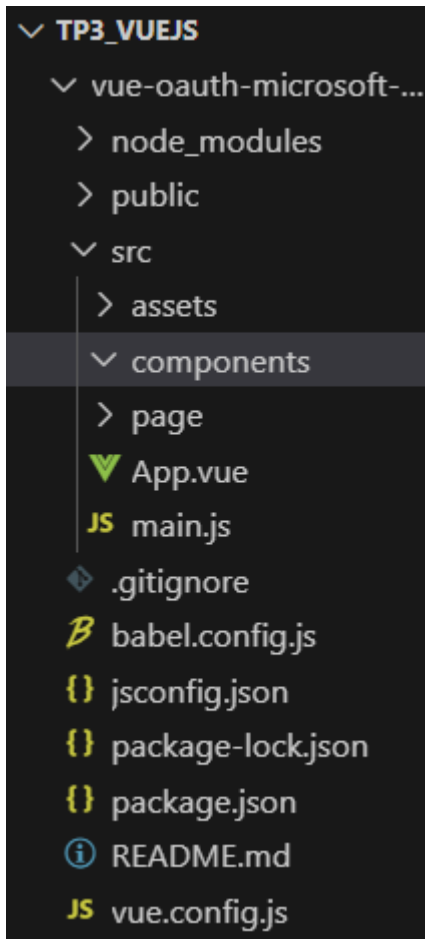
Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

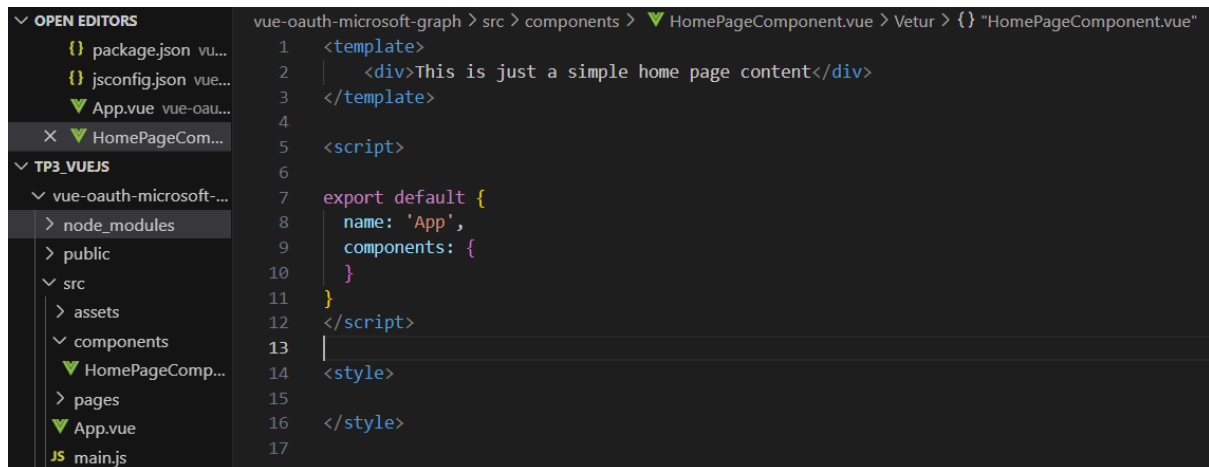
[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)

By default, only the `src/components` folder is intended for storing `.vue` files (aka. vue components). But we are free to use other directories depending on the nature of each component. Let's add `src/pages` to our code base.



Exercise 4: Create the HomePage component inside the right folder. Do not spend too much time on the template content, as it could be a simple sentence. Import it inside App.vue.

Create the new component for “HomePage”



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor on the right. The Explorer sidebar shows the project structure for 'vue-oauth-microsoft-graph', including folders like 'src' and 'components', and files like 'App.vue' and 'main.js'. The 'components' folder is expanded, showing a new file 'HomePageComponent.vue' being created. The Editor shows the content of 'HomePageComponent.vue' with the following code:

```
1 <template>
2   <div>This is just a simple home page content</div>
3 </template>
4
5 <script>
6
7   export default {
8     name: 'App',
9     components: {
10    }
11  }
12 </script>
13
14 <style>
15
16 </style>
17
```

Import this component in the App.vue



The screenshot shows the VS Code interface with the Editor displaying the content of 'App.vue'. The code is as follows:

```
<template>
  <HomePageComponent></HomePageComponent>
</template>

<script>
import HomePageComponent from './components/HomePageComponent.vue';

export default {
  ⚡ name: 'App',
  components: {
    HomePageComponent
  }
}
</script>
```

Exercise 5: Let's begin with the root component, formally App (in src/App.vue). Replace its template with the following content and create the missing components. Add some content to the header (ex. fake home link, fake user name...) and legal credits to the footer. Eventually, polish the looks and feels with scoped CSS.

```
<template>
  <div>
    <base-header>
      <a href="/">Home</a>
      <span>User Name</span>
    </base-header>
    <HomePageComponent />
    <base-footer>
      <div>Legal Credits</div>
    </base-footer>
  </div>
</template>
```

Question 5: What is the difference between scoped and non-scoped CSS?

1. Non-Scoped CSS:

- **Global CSS:** In a typical web project, CSS is global by default. This means that any styles you define can potentially apply to any element on the page, regardless of where the styles are declared. For example, if you define a style for `<h1>`, it will affect all `<h1>` elements throughout the document.
- **No Encapsulation:** Non-scoped CSS does not encapsulate styles within a specific component, so there's a risk of CSS conflicts (e.g., unintended styles overriding each other).

2. Scoped CSS:

- **Component-Scoped:** Scoped CSS ensures that styles are only applied to the component in which they are defined. In Vue.js, for example, you can use the `scoped` attribute within the `<style>` tag to achieve this behavior. This prevents styles from leaking into other components or affecting elements globally.
- **CSS Encapsulation:** Vue handles scoped CSS by adding unique attributes to the elements in the template and generating corresponding styles that target only those elements.

Exercise 6: In order to keep the root component `App` as simple as possible, extract everything related to the layout into a `BaseLayout` component. Using the slot API, allow `BaseLayout` to receive children (to be rendered between the header and the footer).

```
<template>  
  <base-layout>  
    <HomePageComponent />  
  </base-layout>  
</template>
```

Question 6: How behaves non-prop attributes passed down to a component, when its template has a single root element? **Tips:** it is well documented by vue, but you can also try it yourself by passing the `style` attribute with a straight visual effect.

Vue.js allows non-prop attributes (such as `id`, `class`, `style`, etc.) to be passed to child components. If the component has a single root element, those non-prop attributes are automatically applied to the root element of the component. This is known as "attribute inheritance."

For example, if you pass a `style` attribute from a parent component to a child component with a single root element, that `style` attribute will be directly applied to the root element, affecting its visual appearance as if it was directly declared in the template.

This behavior is useful for cases where you don't want to declare all the possible props but still need to pass standard HTML attributes down to components.

Exercise 7: Implement such a `BaseButton`, animated on hover and focus. Do not forget the disabled state. You may try these buttons on your `HomePage` for now.

Here comes the content of the `HomePage`

BaseButton with custom margin

BaseButton disabled

Exercise 8: Add the `color` prop to `BaseButton`. This prop accepts one of 'primary', 'warn' or 'danger' values. It defaults to primary and you should validate the given value matches the enum. Then, dynamically apply styles to the button based on that prop.

Here comes the content of the HomePage

BaseButton with custom margin

BaseButton disabled

BaseButton with color propos

BaseButton with color propos

Father component:

```
<base-header/>
<base-layout>
  <div id="title">Here comes the content of the HomePage</div>

  <div><HomePageComponent color="primary">
    |   BaseButton with custom margin
  </HomePageComponent></div>

  <div><HomePageComponent  color="primary" disabled>
    |   BaseButton disabled
  </HomePageComponent>    </div>

  <div><HomePageComponent  color="warn">
    |   BaseButton with color propos
  </HomePageComponent></div>

  <div><HomePageComponent  color="danger">
    |   BaseButton with color propos
  </HomePageComponent></div>
</base-layout>
<base-footer />
```



Child component:



```
props: {  
  color: {  
    type: String,  
    Required: true  
  }  
},  
  
computed:{  
  buttonColor(){  
    return {  
      "greenBtn":this.color == "primary",  
      "orangeBtn": this.color == "warn",  
      "redBtn": this.color == "danger"  
    }  
  }  
}
```



```
<template>  
  <button :class="buttonColor">  
    <slot></slot>  
  </button>  
</template>
```

```

<style>

.greenBtn{
    background-color:   green;
}

.orangeBtn{
    background-color:   orange;
}

.redBtn{
    background-color:   red;
}

```

Exercise 9: Add a button to the HomePage that is disabled for 2 seconds each time it is clicked. According to the above code, this just means the @click event listener attached to the instance of AsyncComponent instance returns a Promise that waits for 2 seconds before resolving. You can create such a Promise using its constructor and a setTimeout. Also, please write the event handler inside a dedicated method since it is a bit complex.

```

<div><HomePageComponent color="primary" :disabled=isPending @click.stop.prevent=handlerClick>
  Disable and animated for 2 seconds if clicked
</HomePageComponent></div>

```

```

data(){
  return {
    isPending: false
  }
},

```

```

methods:{
  handlerClick(){
    this.isPending = true

    const asyncPromise = new Promise((resolve)=>{
      setTimeout(() => {
        resolve();
      }, 2000);
    })

    asyncPromise.finally(()=> this.isPending = false)
  }
}

```

Before click:

Here comes the content of the HomePage

Disable and animated for 2 seconds if clicked

BaseButton with custom margin

BaseButton disabled

BaseButton with color props

BaseButton with color props

After click:

Here comes the content of the HomePage

Disable and animated for 2 seconds if clicked

BaseButton with custom margin

BaseButton disabled

BaseButton with color propos

BaseButton with color propos

Exercise 10. Change the behaviour of the previous button, so its waiting time increases by one second each it is clicked. Because AsyncButton waits for any promise, whatever how long it takes to resolve, you do not need and you should not change it. Instead, keep trace of the number of clicks in the internal state (data) of the HomePage component (see the counter app example) and use it while forging new promises.

```
<div><HomePageComponent color="primary" :disabled=isPending @click.stop.prevent=handlerClick>  
  Disable and animated for {{counter+1}} seconds if clicked  
</HomePageComponent></div>
```

```
data(){  
  
  return {  
  
    isPending:false,  
    counter : 1  
  }  
},
```

```

methods:{
  handleClick(){
    this.isPending = true

    const asyncPromise = new Promise((resolve)=>{
      setTimeout(() => {
        resolve();
      }, 2000 * this.counter++);
    })

    asyncPromise.finally(()=> this.isPending = false)
  }
}

```

Each click makes waiting time increase one second:

Here comes the content of the HomePage

Disable and animated for 2 seconds if clicked

BaseButton with custom margin

BaseButton disabled

BaseButton with color props

BaseButton with color props

Disable and animated for 3 seconds if clicked

Disable and animated for 4 seconds if clicked

Disable and animated for 5 seconds if clicked

Question 7: Analyse how works the `AsyncButton`. How the child component is aware of the returned Promise by the parent `onClick` handler? When is executed the callback passed to `.finally()`? Why use `.finally()` instead of `.then()`? Etc.

1. The child component becomes aware of the Promise when the parent returns it from the `onClick` handler, likely through prop drilling or similar methods.
2. The callback passed to `.finally()` is executed after the Promise settles, regardless of whether it was resolved or rejected.
3. `.finally()` is chosen over `.then()` when you need to run cleanup tasks that should happen regardless of the Promise outcome, leading to cleaner and more efficient code.

Question 8: Which bug is introduced if `inheritAttrs: false` is missing or set to `true` in `AsyncButton`? Why?

In Vue, setting `inheritAttrs: false` prevents any additional attributes (like `onClick` handlers, classes, etc.) from automatically being passed to the root element of a child component. This can be particularly important when dealing with components like `BaseButton` or `AsyncButton`.

Without `inheritAttrs: false` (or if set to `true`): Attributes from the parent, such as `onClick` or classes, are applied to the child component's root element. This

could cause problems like duplicate event triggers (e.g., two `onClick` handlers firing) or the application of unintended attributes, leading to unpredictable behaviors.

In the case of `AsyncButton`: The `onClick` event is managed within the `handleClick()` method. If `inheritAttrs: false` is not specified, the parent's `onClick` could also be bound to `BaseButton`, causing the event to be handled twice. By setting `inheritAttrs: false`, you avoid this, ensuring that only the intended behavior (the internal `handleClick()` logic) is executed, while preventing unnecessary attributes from being passed down. This gives better control over how child components handle attributes, minimizing conflicts.