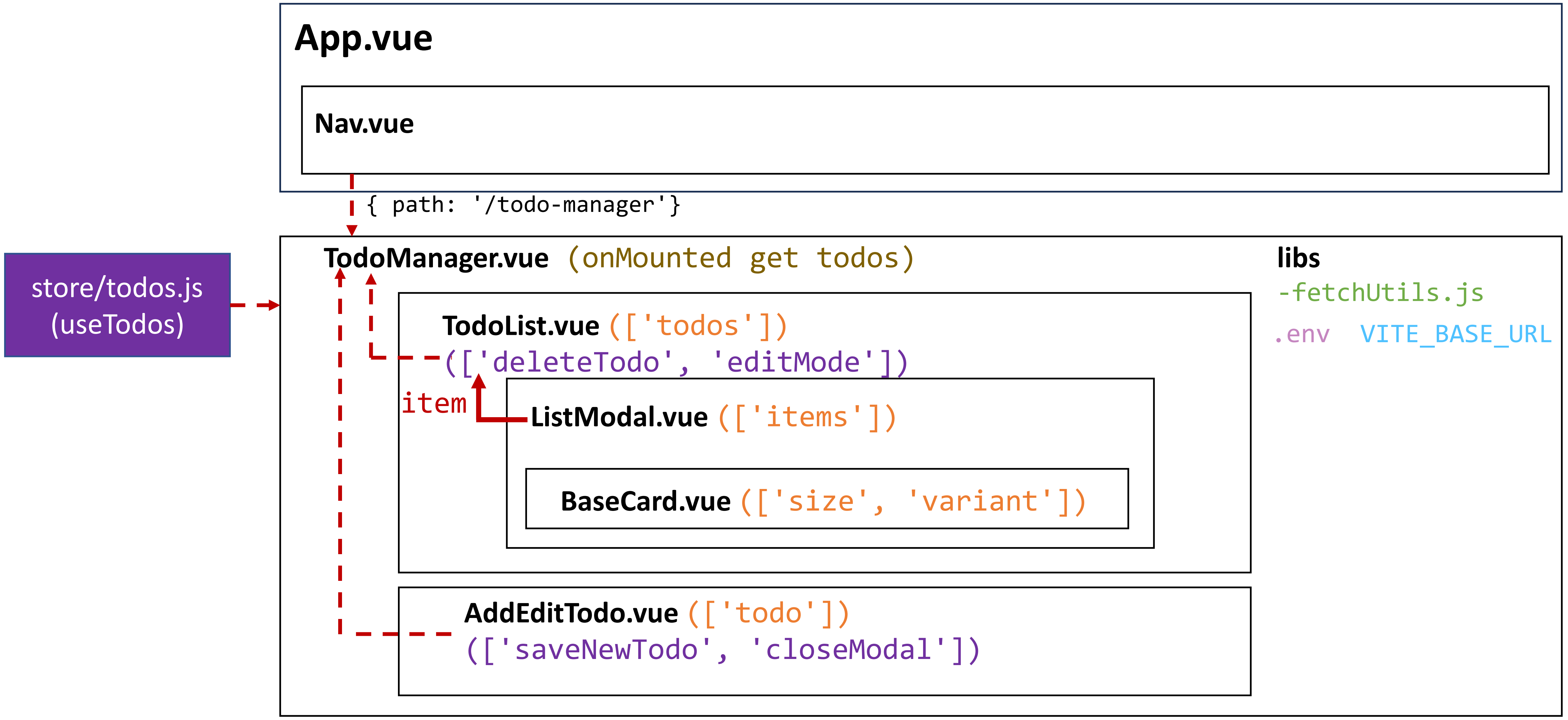
Chapter 9 – Basic State Management

Asst.Prof. Dr. Umaporn Supasitthimethee
ผศ.ดร.อุมาพร สุภสิทธิเมธี

[1] https://vuejs.org/guide/scaling-up/state-management.html

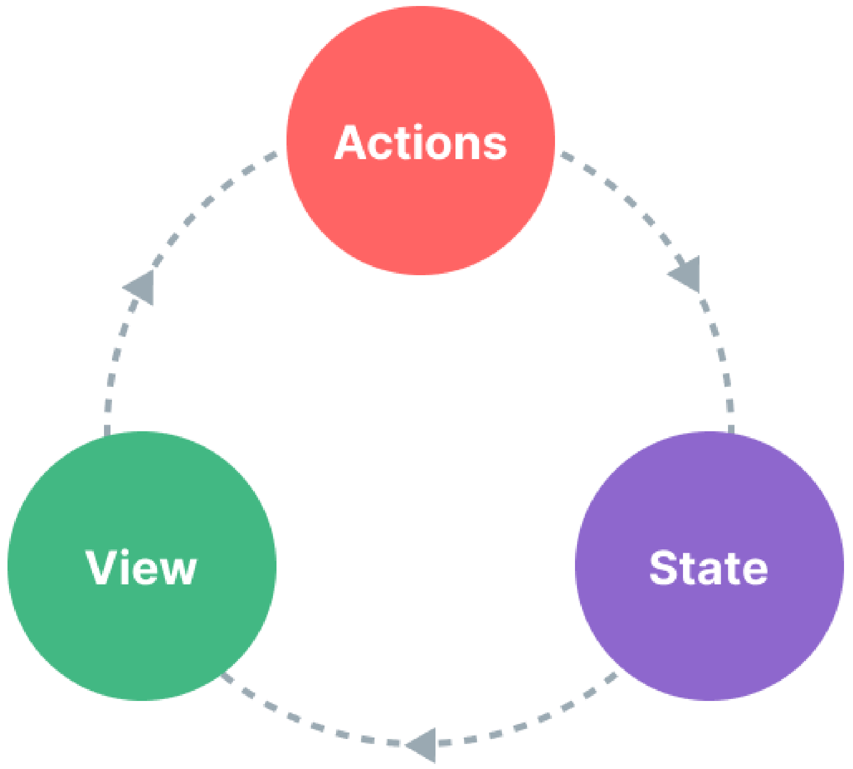
[2] https://pinia.vuejs.org/

TodoList Design Diagram


App.vue
Nav.vue
{ path: '/todo-manager'}
store/todos.js (useTodos)
TodoManager.vue (onMounted get todos)
libs
-fetchUtils.js
.env VITE_BASE_URL
TodoList.vue (['todos'])
(['deleteTodo', 'editMode'])
item
ListModal.vue (['items'])
BaseCard.vue (['size', 'variant'])
AddEditTodo.vue (['todo'])
(['saveNewTodo', 'closeModal'])

What is State Management?

- It is a self-contained unit with the following parts:
  - The **state**, the source of truth that drives our app;
  - The **view**, a declarative mapping of the state;
  - The **actions**, the possible ways the state could change in reaction to user inputs from the view.



Handle Your State Management (without pinia)

src/App.vue

```
<script setup>
import { store } from './data/counter.js'
import ShowCounter from './components/ShowCounter.vue'
</script>

<template>
  <div>
    App counter: {{ store.counter }}
    <button @click="store.increment()">+</button>
    <button @click="store.decrement()">-</button>

    <div>
      <ShowCounter />
    </div>
  </div>
</template>

<style></style>
```

src/data/counter.js

```
import { reactive } from 'vue'
export const store = reactive({
  counter: 1,
  increment() {
    this.counter++
  },
  decrement() {
    this.counter--
  }
})
```

src/components/ShowCounter.vue

```
<script setup>
import { store } from '../data/counter.js'
</script>

<template>
  <div>Show Counter: {{ store.counter }}</div>
</template>
```

Why needs State Management

- However, the simplicity starts to break down when we have multiple components that share a common state:
  1. Multiple views may depend on the same piece of state.
  2. Actions from different views may need to mutate the same piece of state.

these are brittle and quickly lead to unmaintainable code.

A simpler and more straightforward solution is to extract the shared state out of the components, and manage it in a global singleton.

Pinia

- Pinia is a state management library that supports in large-scale production applications.
- It is maintained by the Vue core team and works with both Vue 2 and Vue 3.

https://vuejs.org/guide/scaling-up/state-management.html#pinia

Why should I use Pinia?

- Pinia is a store library for Vue, it allows you to share a state across components/pages in large-scale production applications
- Existing users may be familiar with Vuex, the previous official state management library for Vue.
- With Pinia serving the same role in the ecosystem, Vuex is now in maintenance mode. It still works but will no longer receive new features. It is recommended to use Pinia for new applications.
- Compared to Vuex, Pinia provides a simpler API with less ceremony, offers Composition-API-style APIs, and most importantly, has solid type inference support when used with TypeScript.

What is a Store?

- A Store (like Pinia) is an entity holding state and business logic that isn't bound to your Component tree.
- In other words, **it hosts global state**. It's a bit like a component that is always there and that everybody can read off and write to.
- It has **three concepts**, the *state*, *getters* and *actions* and it's safe to assume these concepts are the equivalent of *data*, *computed* and *methods* in components.

When should I use a Store?

- A store should contain data that can be accessed throughout your application.
- This includes data that is used in many places, e.g. User information that is displayed in the navbar, as well as data that needs to be preserved through pages, e.g. a very complicated multi-step form.
- On the other hand, you should avoid including in the store local data that could be hosted in a component instead, e.g. the visibility of an element local to a page.
- Not all applications need access to a global state, but if yours need one, Pinia will make your life easier.

Install pinia with your favorite package manager

```
> npm install pinia
```

https://pinia.vuejs.org/getting-started.html#installation

Create a pinia (the root store) and pass it to the app:

main.js

```
import { createApp } from 'vue'
import App from './App.vue'
import { createPinia } from 'pinia'
createApp(App).use(createPinia()).mount('#app')
```

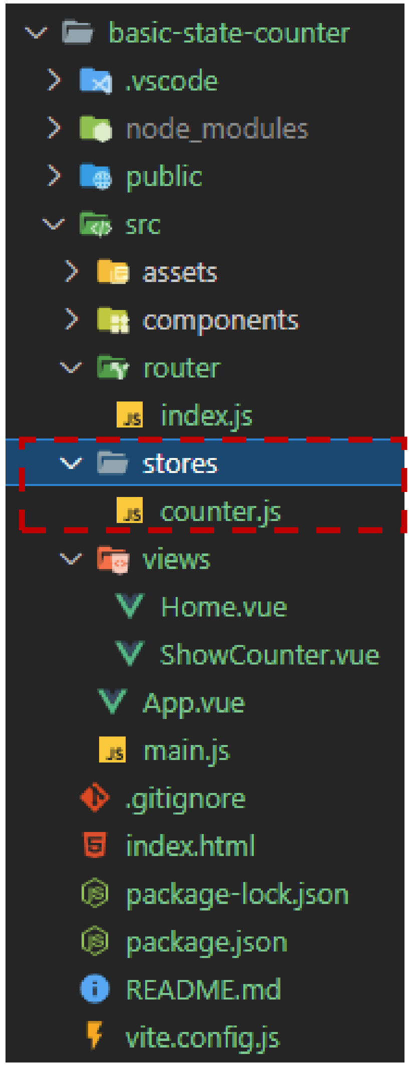
Defining Your Global Store Data

In *Setup Stores*:
- ref() become state properties
- computed() become getters
- function() become actions

```
import { defineStore } from 'pinia'
// useStore could be anything like useUser, useCart
// the first argument is a unique id of the store across your application
export const useStore = defineStore('main', {
// other options...
})
```

src/stores/counter.js

```
import { defineStore, acceptHMRUpdate } from 'pinia'
import { ref, computed } from 'vue'
export const useCounter = defineStore('counter', () => {
  const count = ref(1)
  const doubleCount = computed(() => count.value * 2)
  const resetCount = () => (count.value = 0)
  const addCounter = () => (count.value += 1)
  return { count, doubleCount, resetCount, addCounter }
})

if (import.meta.hot) {
  import.meta.hot.accept(acceptHMRUpdate(useCounter, import.meta.hot))
}
```

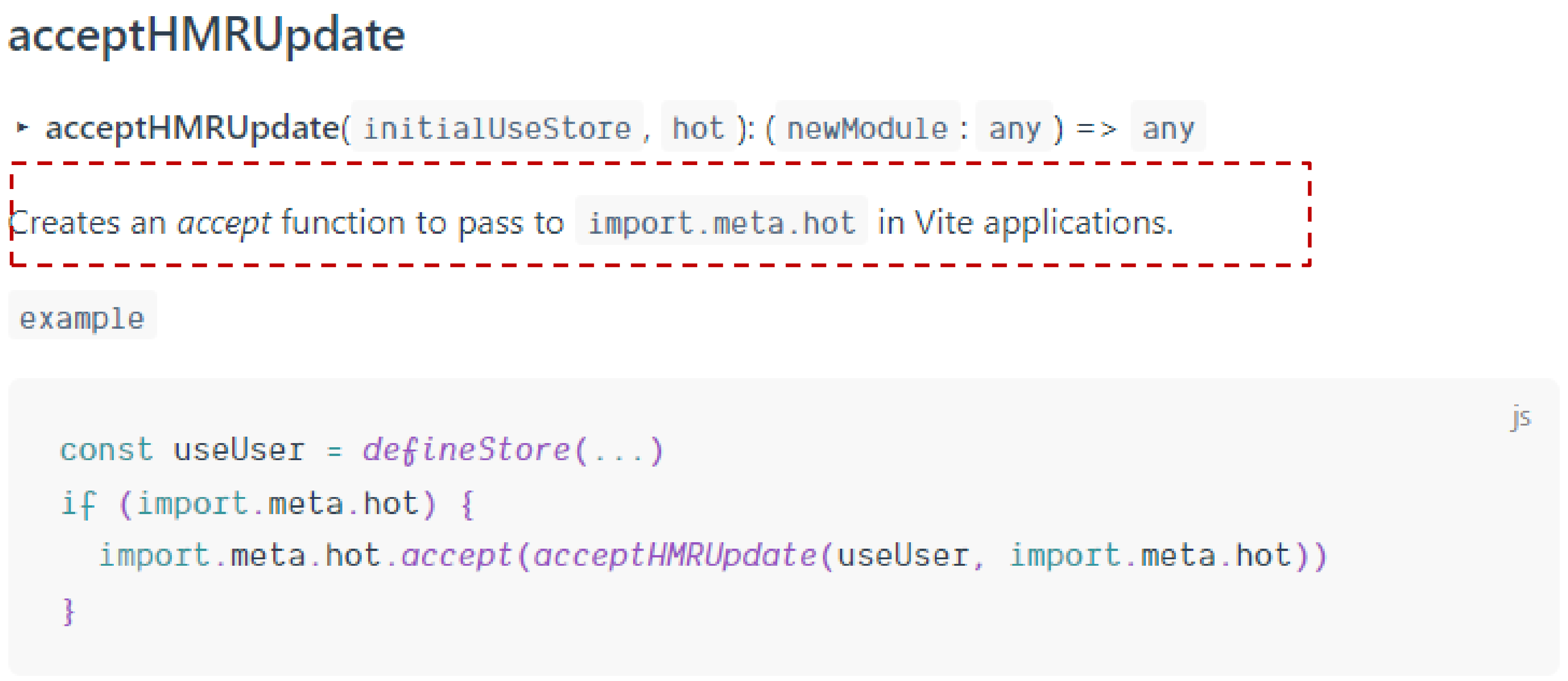


```
basic-state-counter
  .vscode
  node_modules
  public
  src
    assets
    components
    router
      index.js
    stores
      counter.js
    views
      Home.vue
      ShowCounter.vue
    App.vue
    main.js
  .gitignore
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js
```

https://pinia.vuejs.org/core-concepts/#defining-a-store

acceptHMRUpdate

Pinia supports Hot Module replacement so you can edit your stores and interact with them directly in your app without reloading the page, allowing you to keep the existing state, add, or even remove state, actions, and getters.



acceptHMRUpdate

▸ acceptHMRUpdate(`initialUseStore` , `hot`): (`newModule` : `any`) => `any`

Creates an *accept* function to pass to `import.meta.hot` in Vite applications.

`example`

```js
const useUser = defineStore(...)
if (import.meta.hot) {
  import.meta.hot.accept(acceptHMRUpdate(useUser, import.meta.hot))
}
```

you will have to add (and adapt) `if(import.meta.hot) {}` after the creation of the *store definition*

https://pinia.vuejs.org/api/modules/pinia.html#accepthmrupdate

Use Your Global Store Data

Home.vue

```
<script setup>
import { useCounter } from '../stores/counter.js'

const myCounter = useCounter()
</script>

<template>
  <div>
    <h1>Home Page</h1>
    Counter (Home.vue): {{ myCounter.count }}

    <div>
      <button @click="myCounter.addCounter()">Add 1 to Counter</button>
      <button @click="myCounter.resetCount()">Reset</button>
    </div>
  </div>
</template>

<style></style>
```

`storeToRefs` Function

- In order to extract properties from the store while keeping its reactivity, you need to use `storeToRefs()`.
- It will create `refs` for every reactive property. This is useful when you are only using state from the store but not calling any action.
- Note you can destructure actions directly from the store as they are bound to the store itself too:

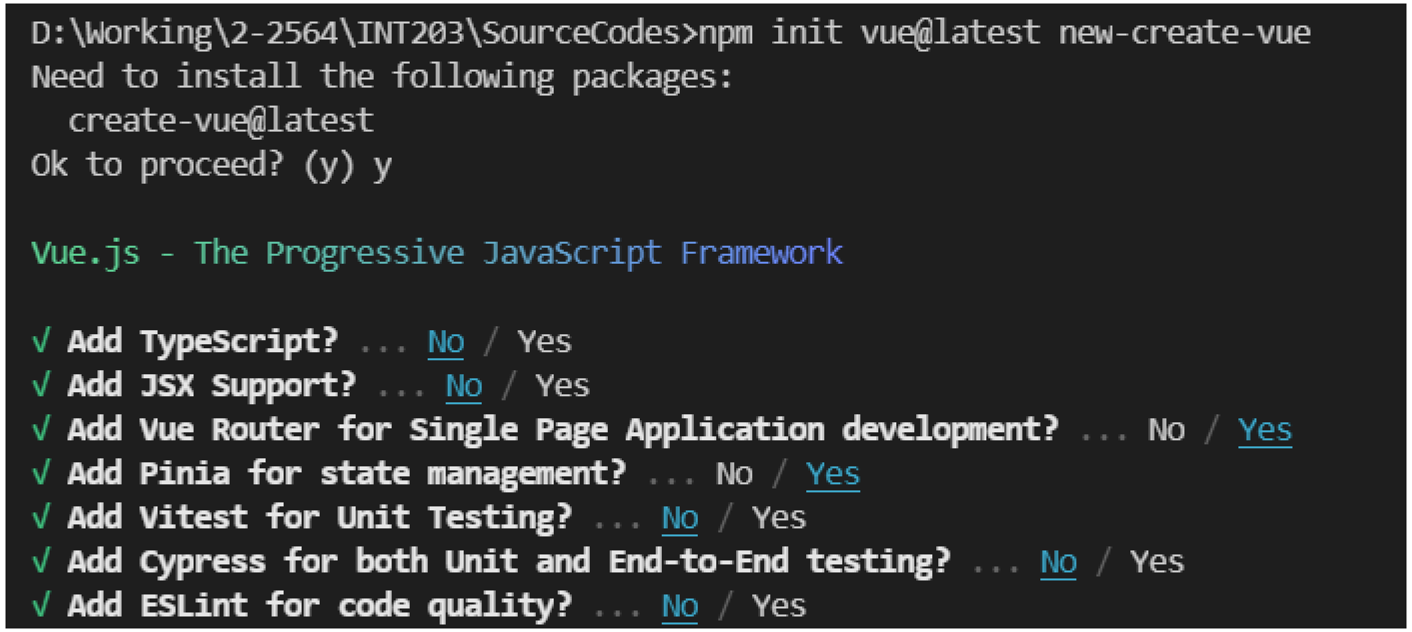
https://pinia.vuejs.org/api/modules/pinia.html#storetorefs

Use Your Global Store Data

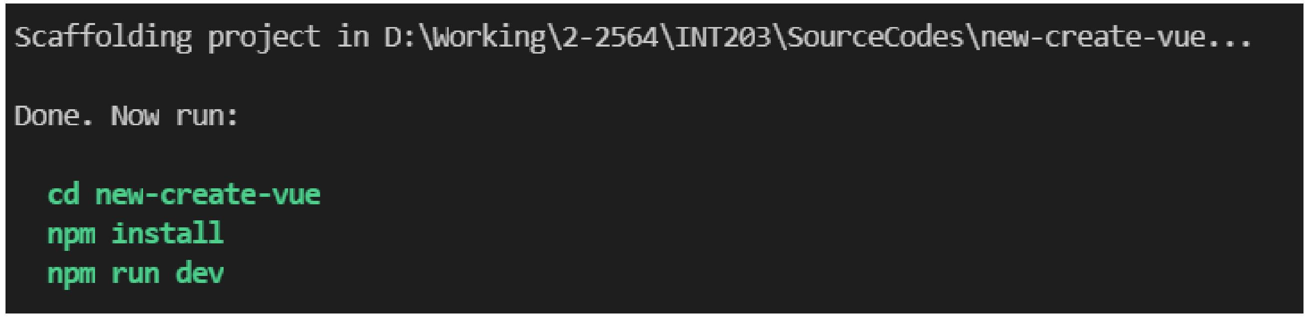
ShowCounter.vue

```
<script setup>
import { useCounter } from '../stores/counter.js'
import { storeToRefs } from 'pinia'
const myCounter = useCounter()

//extract properties from the store while keeping its reactivity
const { count, doubleCount } = storeToRefs(myCounter)

//can destructure method actions directly from the store
const { addCounter, resetCounter } = myCounter
</script>

<template>
  <div>
    <h1>Show Counter Page</h1>
    <p>Counter (ShowCounter.vue): {{ count }}</p>
    <p>DoubleCounter (ShowCounter.vue): {{ doubleCount }}</p>
    <button @click="resetCounter">RESET</button>
  </div>
</template>
```

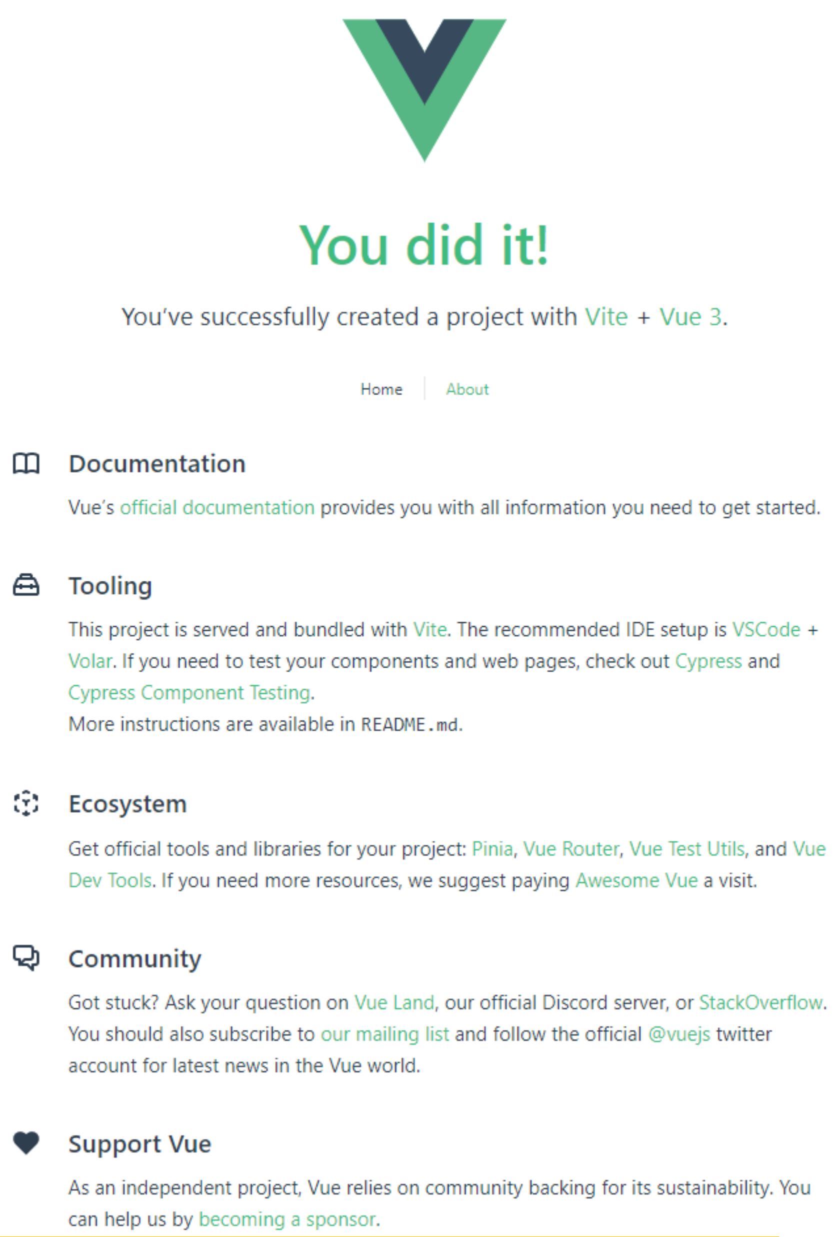
create Vue project (include pinia, vue router)



```
D:\Working\2-2564\INT203\SourceCodes>npm init vue@latest new-create-vue
Need to install the following packages:
  create-vue@latest
Ok to proceed? (y) y

Vue.js - The Progressive JavaScript Framework

√ Add TypeScript? ... [✓ No] / Yes
√ Add JSX Support? ... [✓ No] / Yes
√ Add Vue Router for Single Page Application development? ... No / [✓ Yes]
√ Add Pinia for state management? ... No / [✓ Yes]
√ Add Vitest for Unit Testing? ... [✓ No] / Yes
√ Add Cypress for both Unit and End-to-End testing? ... [✓ No] / Yes
√ Add ESLint for code quality? ... [✓ No] / Yes
```



```
Scaffolding project in D:\Working\2-2564\INT203\SourceCodes\new-create-vue...

Done. Now run:

  cd new-create-vue
  npm install
  npm run dev
```



You did it!

You've successfully created a project with Vite + Vue 3.

Documentation

Vue's official documentation provides you with all information you need to get started.

Tooling

This project is served and bundled with Vite. The recommended IDE setup is VSCode + Volar. If you need to test your components and web pages, check out Cypress and Cypress Component Testing.

More instructions are available in `README.md`.

Ecosystem

Get official tools and libraries for your project: Pinia, Vue Router, Vue Test Utils, and Vue Dev Tools. If you need more resources, we suggest paying Awesome Vue a visit.

Community

Got stuck? Ask your question on Vue Land, our official Discord server, or StackOverflow. You should also subscribe to our mailing list and follow the official @vuejs twitter account for latest news in the Vue world.

Support Vue

As an independent project, Vue relies on community backing for its sustainability. You can help us by becoming a sponsor.

Vue3 Ecosystem

Vue 3 Ecosystem Maturing
Nuxt
Vuetify
Quasar
VueUse
NaiveUI
PrimeVue

Nuxt - Empower Vue developers to ship great softwares with confidence. Help you ship fullstack Vue apps that are performant and SEO friendly.

Vuetify - Vuetify is a no design skills required UI Library with beautifully handcrafted Vue Components.

Quasar - Front-end framework with VueJS components. Get a state-of-the-art UI for your websites and apps. Best support for desktop and mobile browsers.

VueUse - is a collection of utility functions based on Composition API.

Naive UI - provides some tools for developers to create themed components easier.

PrimeVue - Next Generation Vue UI Component Library

https://github.com/vuejs/awesome-vue.git