



# **Pinia**

**State Management Solution for Vue**

**Ye Myint Soe (Salai)**

Hornbill Technology

## မာတိကာ

မိတ်ဆက်	3
အခန်း (၁) – What is Pinia?	4
အခန်း (၂) – NodeJS Installation for Node Package Manager (NPM)	6
အခန်း (၃) – Vue Installation Using CLI	8
Install Vue	
Create a Vue Project	
အခန်း (၄) – Pinia Installation & Registration	12
Install the Pinia	
Register the Pinia	
အခန်း (၅) – Store Creation & Implementation	13
What is a Store?	
Option Store	
state	
Accessing the Store from the Component	
getters	
actions	
Accessing a Store from Multiple Components	
Setup Store	
အခန်း (၆) – Pinia Blog App Using Vue3, Pinia & Bootstrap5	29
နိဂုံးချုပ်	38

## မိတ်ဆက်

Pinia သည် Vue Router ကိုဖန်တီးခဲ့တဲ့ Vue Core အဖွဲ့ဝင် **Eduardo San Martin Morote** က ဖန်တီးခဲ့ပါတယ်။ Pinia သည် ဒီစာအုပ်လေး ရေးနေချိန်မှာ Version 2 ဖြစ်ပြီး Vue ကတော့ Version 3 ဖြစ်ပါတယ်။ ဒါပေမယ့် Pinia က Vue2 နဲ့ Vue3 ၂ခုစလုံးမှာ အသုံးပြုလို့ရပါတယ်။ Pinia ဟာ Vue Application တွေကို သာမက Vue Developer တွေကိုပါ Speed Up လုပ်ပေးနိုင်တဲ့ State Management Solution တစ်ခုဖြစ်ပါတယ်။ Vue Developer တွေက Vue Application တွေကို ဖန်တီးတဲ့ အခါမှာ Pinia လို State Management Library ကိုအသုံး မပြုဘဲ တည်ဆောက်နေရင် Application တစ်ခုလုံး Data Passing နဲ့ Component Event တွေနဲ့ ရှုပ်ပွနေမှာ အမှန်ပဲဖြစ်ပါတယ်။

Vue Version 3 မတိုင်ခင်မှာ ဆိုရင် Pinia ကို မသုံးဘဲ Vuex ဆိုတဲ့ State Management Library တစ်ခုကို အသုံးပြုခဲ့ကြပါတယ်။ ဒါပေမဲ့ Vue Version 3 ရောက်လာပြီးတဲ့ နောက်ပိုင်းမှာတော့ Vue က Pinia ကို သူ့ရဲ့ Default State Management Library အဖြစ် ကြေညာလိုက် ပါတယ်။ ဒီလို အကျိုးဖြစ် လာခြင်းရဲ့ အကြောင်းတရား တွေလည်းရှိနေလို့ပါ။ Pinia ဟာ Vuex ထက် အရမ်းရိုးရှင်းပြီး နားလည်ရလည်း လွယ်ကူပါတယ်။ Vuex မှာ ပါခဲ့တဲ့ Step တွေကို Step အနည်းဆုံး ဖြစ်လာအောင် လျှော့ချထားပါတယ်။ Pinia နဲ့ Store တွေ ဖန်တီးရာမှာ နည်းလမ်း ၂ မျိုးနဲ့ ဖန်တီးနိုင်လို့ ကိုယ်ကြိုက်တဲ့ နည်းကိုအသုံးပြု လို့ရပါတယ်။ နည်းလမ်း ၂ မျိုးထဲမှ တစ်မျိုးဟာဆိုရင် Vue3 မှာပါလာတဲ့ Composition API ရေးထုံးနဲ့ တစ်ပုံစံတည်း တူနေလို့ကို Vue3 Composition API လေ့လာထားသူတွေ အတွက်တော့ မျက်စိမှိတ်ပြီး ရေးရုံပါပဲ။ ဒီနေရာမှာ Pinia နဲ့ Vuex ကို Comparing လုပ်နေတာ မဟုတ်တော့ ဒီမှာပဲ ရပ်ထားပါရစေ။

Pinia ရဲ့ အသေးစိတ် အကြောင်းအရာ ကိုတော့ နောက် အခန်းကနေ ဆက်လေ့လာကြပါမယ်။ **Pinia ကို မလေ့လာမီမှာ မိတ်ဆွေတို့သိထားရမှာ တစ်ခုရှိပါတယ်။ အဲဒါက ဘာလဲဆိုတော့ Pinia သည် VueJS အတွက် ဖန်တီးထားတဲ့ Library တစ်ခုဖြစ်တဲ့အတွက်ကြောင့် ဒီ Pinia ကိုမလေ့လာမီမှာ မိတ်ဆွေတို့က VueJS ကို အသင့်သင့်သိထားရမယ်ဖြစ်ပါတယ်။ ဒီ စာအုပ်ထဲမှာတော့ VueJS Version 3 Composition API ကိုအသုံးပြုပြီး သင်ပြ ပေးသွားမယ် ဖြစ်ပါတယ်။**

## အခန်း (၁) – What is Pinia?

### What is Pinia?

Pinia ဆိုတာ VueJS ရဲ့ Store Library (သို့) State Management Solution Library တစ်ခုဖြစ်ပါတယ်။ Pinia ကို အသုံးပြု၍ Global State တွေ တည်ဆောက်ပြီး အဲ့ဒီ State တွေကို Components တွေ Pages တွေသို့ Share လုပ်လို့ရပါတယ်။ ဆိုလိုတာက Multiple Components တွေကနေ Pinia Store ထဲမှာ ရှိနေတဲ့ data တွေကို ရယူလို့ရတယ်၊ ပြုပြင်ပြောင်းလဲ လို့ရတယ်။

ဆိုကြပါစို့ ကျွန်တော်တို့မှာ Vue app တစ်ခုရှိတယ်၊ အဲ့ဒီ Vue app မှာ ComponentA, ComponentB နဲ့ ComponentC ဆိုပြီး Component ၃ ခုရှိတယ် ။ ComponentB နဲ့ ComponentC ကို ComponentA မှာ import လုပ်ပြီး အသုံးပြုထားတယ်ဆိုပါစို့။

ComponentA မှာ name: “John Doe” ဆိုပြီး data တစ်ခုရှိတယ်။ အဲ့ဒီ data ကိုသုံး၍ Component မှာ Display လုပ်ထားသလို ComponentB နဲ့ ComponentC မှာလည်း Display လုပ်ချင်တယ်ဆိုရင် ... အဲ့ဒီ Component B, C ၊ ခုလုံးကို v-bind နဲ့ name data ကို Component Tag ကနေ pass လုပ်ပေးရတယ်။

Parent ComponentA ကနေ Pass လုပ်လိုက်တဲ့ data ကို Child Component B နဲ့ C ကနေ props တွေသုံးပြီး Access လုပ်ပြီး Display ပြရမယ်ဖြစ်တယ်။ ပြီးရင် နောက်ထပ် လုပ်ချင်တာ တစ်ခုရှိ တယ်ဆိုပါစို့။

ComponentB ကနေ name data ကို Change လိုက်ရင် Component A နဲ့ C ၊ ခုစလုံးမှာ Effect ဖြစ်ရမှာ ဖြစ်သလို ComponentC ကနေ name data ကို Change လိုက်ရင်လည်း Component A နဲ့ B ၊ ခုစလုံးမှာ effect ဖြစ်လာရမယ်ပေါ့။

name data က Parent ComponentA မှာပဲရှိနေလို့ Parent ComponentA က data ကို Change လုပ်နိုင်မှ Component ၃ ခုလုံးမှာ Effect ဖြစ်သွားမှာပါ။ ဆိုတော့ Child Component တွေက နေ Component Event ဆိုတဲ့ Feature ကိုသုံးပြီး Event တစ်ခုကို name data နဲ့ အတူ Fire လုပ်ရမယ်။ ပြီးတော့မှ Parent ComponentA ကနေ အဲ့ဒီ Event ကို Access လုပ်ပြီး name data ကို Update လုပ်ပေးရတယ်။

ကဲ ကြည့်ပါအုံး Step တွေက များပြီး Code ရေးရတာ လက်ဝင်တယ်။ Project အသေးတွေ ရေးနေရင် မပြောလိုဘူး၊ ဒါပေမယ့် Project အကြီးတွေသာ ရေးရမယ်ဆိုရင် ဘယ်လိုမှ Component Event နဲ့ ဖြေရှင်းလို့မရပါဘူး။ ဒါကြောင့် အဲဒီလို ရှုပ်ထွေးတဲ့ ပြဿနာကို ဖြေရှင်းဖို့ State Management Solution ဖြစ်တဲ့ Pinia ကိုသုံးပေးရတာ ဖြစ်ပါတယ်။

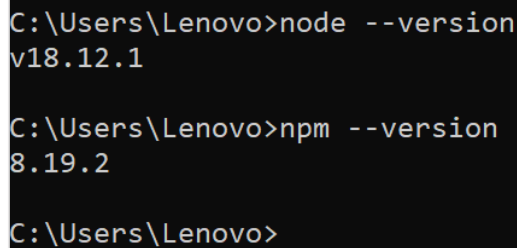
## အခန်း (၂) – NodeJS Installation for Node Package Manager (NPM)

Pinia (State Management Library) ကို Vue Counter app လေးတစ်ခု ဖန်တီးပြီး လေ့လာကြည့်ရအောင်။ Vue app ကို Install လုပ်လိုရတဲ့ နည်းအမျိုးမျိုး ရှိတဲ့ ထဲက CLI ကိုအသုံးပြုပြီး Install လုပ်ပြပါမယ်။

ပထမဦးဆုံး Vue ကို Command Line Interface (CLI) နဲ့ Install လုပ်မယ်ဆိုရင် NodeJS ရှိထားဖို့တော့လိုပါတယ်။ ကိုယ့်စက်ထဲမှာ Node ရှိမရှိကို cmd ဖွင့်ပြီး ဒီလိုလေး ရိုက်ပေးပါ။

```
node --version (or) node -v
```

```
npm --version (or) npm -v
```



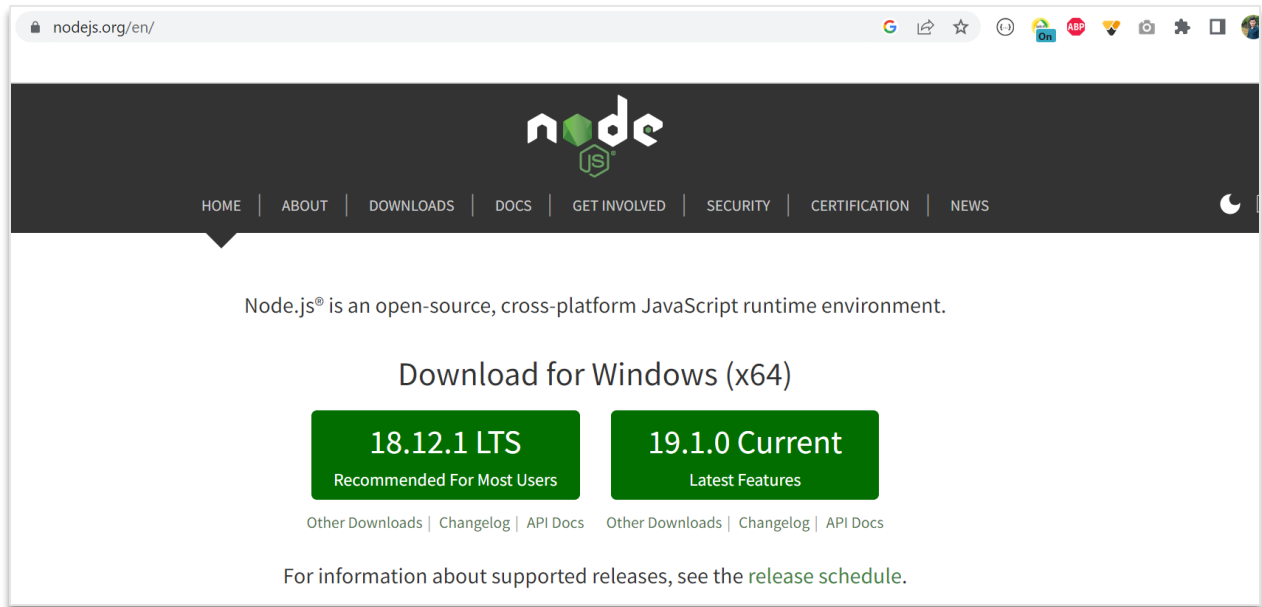
```
C:\Users\Lenovo>node --version
v18.12.1

C:\Users\Lenovo>npm --version
8.19.2

C:\Users\Lenovo>
```

အပေါ်မှာပြသလို Node နဲ့ NPM Version တို့ကိုတွေ့ရရင် NodeJS ကို Installed လုပ်ထားလို့ပါ။ မတွေ့ရရင်တော့ Vue app ကို CLI နဲ့ Install ပြုလုပ်လို့မရသေးပါဘူး။ ဒါကြောင့် NodeJS ကိုဒီ <https://nodejs.org/en> Website ကိုသွားပြီး Download သွားလုပ်ပေးပါ။

ဒီ <https://nodejs.org/en> Website ကိုသွားရင် အောက်မှာပြထားတဲ့နေရာ ရောက်သွားပါမယ်။ အဲ့ဒီ ထဲကမှ ဘယ်ဘက်က LTS ဆိုတဲ့ဟာကို Click နှိပ်ပြီး Download ရယူပါ။ ပြီးရင် မိမိစက်ထဲ ကို Install လုပ်ပေးပါ။ Installation အောင်မြင်ပြီး Node နဲ့ NPM Version ကို ပြန်စစ်ကြည့်လို့ Version နံပါတ် ပေါ်လာရင် ကျွန်တော်တို့ ဆက်သွားလို့ရပါပြီ။



## အခန်း (၃) – Vue Installation Using CLI

### Step 1. Install Vue

Vue ကို Install လုပ်ဖို့ အောက်က Command Line ကို Run ပေးပါ။

```
D:\vue>npm install -g @vue/cli
```

အပေါ်က Command Line ကဘာကို ဆိုလိုသလဲဆိုရင် VueJS ကို npm နဲ့ Globally ဒီစက်ထဲမှာ Install ပြုလုပ်မယ်လို့ ပြောချင်တာပါ။ Installation Process ပြီးသွားရင် Vue Version ကိုအောက်ပါအတိုင်း စစ်ကြည့်လို့ရပါတယ်။

```
D:\vue>vue --version
@vue/cli 5.0.8

D:\vue>
```

### Step 2. Create a Vue Project

Vue app ကို CLI နဲ့ Install လုပ်လို့ ရပြီမို့ ကိုယ် Install လုပ်ချင်တဲ့ Folder ထဲမှာ cmd ဖွင့်ပြီး အောက်က Command Line နဲ့ Project နာမည်ကိုပါ တစ်ခါတည်းပေးပြီး Run ပေးပါ။

```
D:\vue>vue create pinia-counter
```

အပေါ်က Command Line ကို Run ပေးပြီးရင် Install လုပ်ပေးမဲ့ Version ကိုပြလာပါမယ်။ Default ([Vue 3] babel, eslint) ဆိုတဲ့ ဟာကိုရွေးပေးပါ။ ပြီးရင် Successfully ဆိုတဲ့ Message နဲ့အတူ လုပ်ဆောင်ရမယ့် အချက် ၂ ချက်ကို အောက်ပါအတိုင်းပြပေးပါတယ်။



```

? ?? Successfully created project pinia-counter.
? ?? Get started with the following commands:

$ cd pinia-counter
$ npm run serve

D:\vue>

```

အပေါ်က အချက် ၂ ချက်က ဘာကိုပြော ချင်သလဲ ဆိုတော့ cd (change directory) နဲ့ pinia-counter ဆိုတဲ့ Project Folder ထဲကို သွားပေးရန်၊ ပြီးရင် npm run serve ဆိုပြီး pinia-counter VueJS app ကို Serve လုပ်ပေးရန် ဖြစ်ပါတယ်။ npm run serve မလုပ်မီမှာ code . နဲ့ Project Folder ကို VS Code နဲ့ဖွင့်ထားပေးပါ။ ပြီးမှ ပြန် Run ပေးပါ။ အောက်ပါအတိုင်း ဖြစ်လာပါမယ်။

```

DONE Compiled successfully in 33137ms

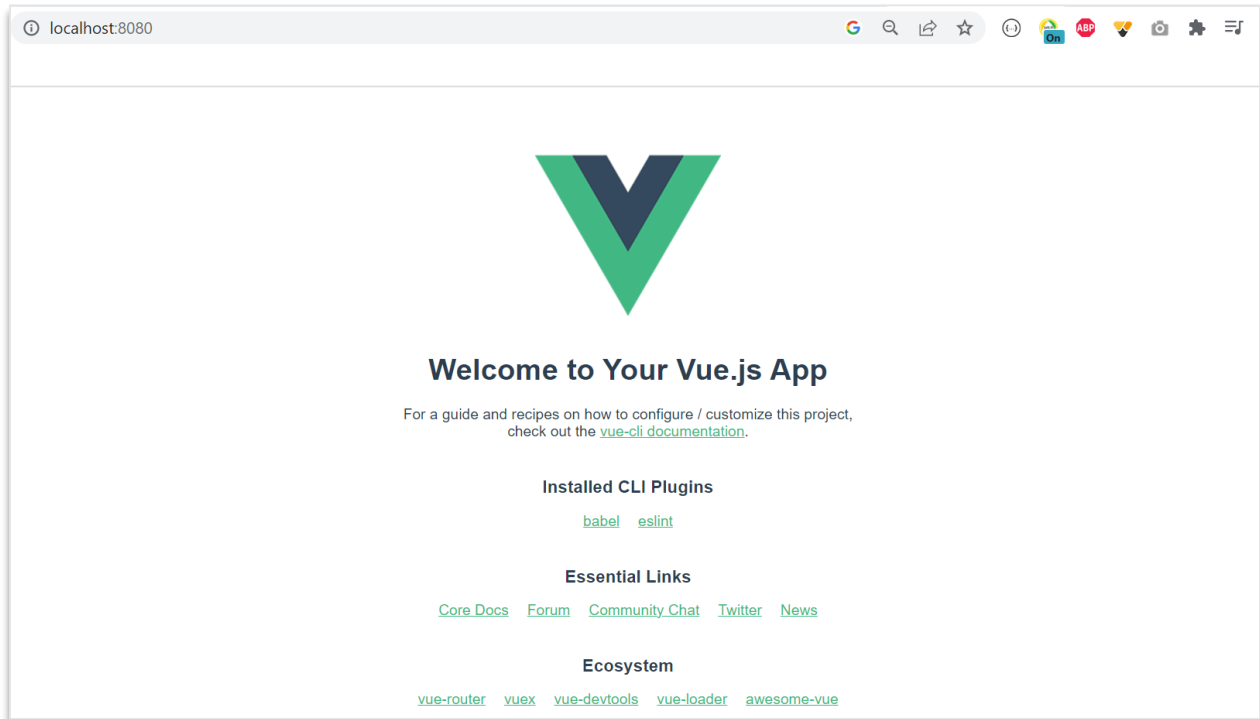
App running at:
- Local:   http://localhost:8080/
- Network: http://192.168.100.11:8080/

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

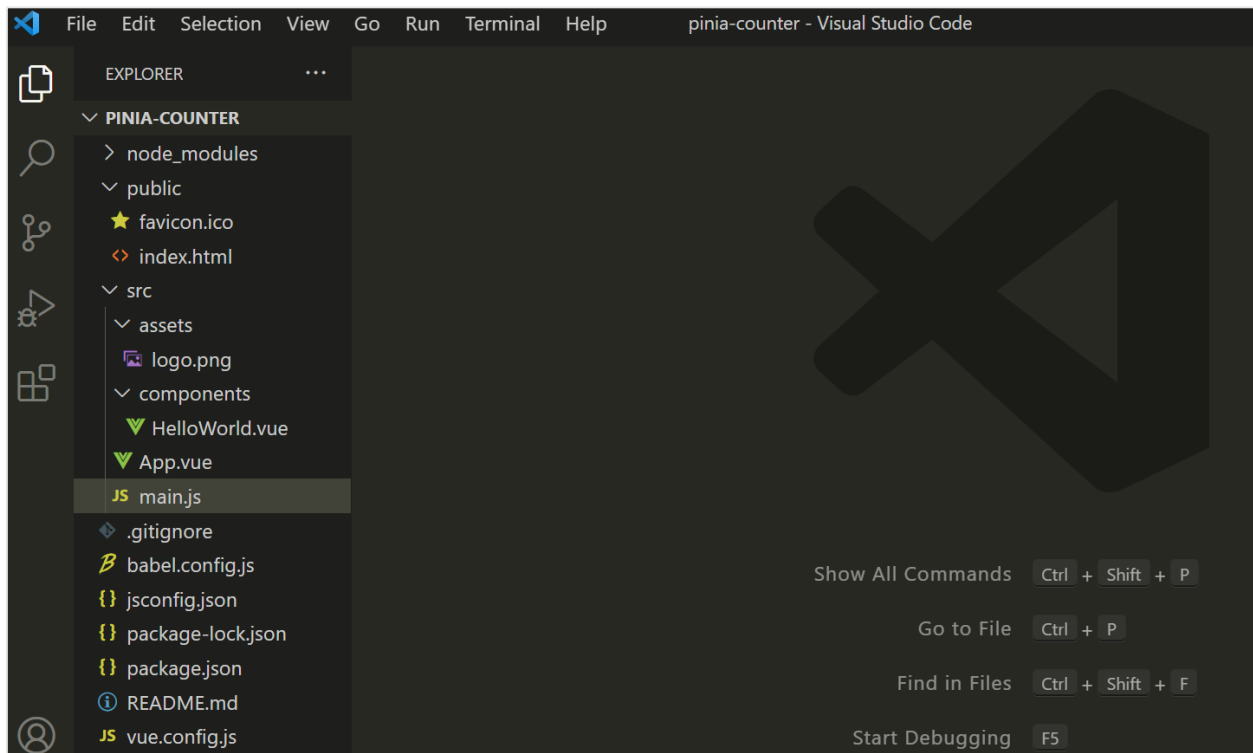
```

အပေါ်က ပုံမှာ Local ရယ် Network ရယ်ဆိုပြီး Project Run လို့ရမယ့် Address ၂ ခု ပေးထားတာ တွေ့ရမှာပဲ ဖြစ်ပါတယ်။ ကိုယ့်ကြိုက်တဲ့ တစ်ခုကို Web Browser မှာသွား Run ပေးပါ။ အောက်ပါတိုင်းပေါ်လာပါမယ်။

## Pinia – State Management Solution for Vue



အခုဆိုရင် Vue Counter app တစ်ခုကို အောင်အောင် မြင်မြင်နဲ့ Install လုပ်နိုင်ခဲ့ပါပြီ။ VS Code မှာ ဖွင့်ထားတဲ့ Project ကိုသွားကြည့်ရင်လည်း အောက်ပါအတိုင်း မြင်ရမှာပါ။



အပေါ်မှာ ပြထားတဲ့ Project Folder ထဲကမှ မလိုအပ်တာတွေကို ဖျက်ပစ်ပါ။ နောက်မှ လိုအပ်လာရင်လည်း ပြန်ဆောက်ပေးလို့ရပါတယ်။ ဖျက်ရမယ့် Files နဲ့ Folder List တွေကတော့ -

- public/favicon.ico
- src/assets
- components/HelloWorld.vue

တို့ဖြစ်ပါတယ်။

## အခန်း (၄) – Pinia Installation & Registration

### Step 1. Install the Pinia

ကဲ ဒါဆို Vue Counter app ထဲမှာ Pinia – State Management Library ကို Install လုပ်ကြရအောင်။  
Install လုပ်ဖို့ရာ အောက်ပါ Command Line ကို Run ပေးပါ။

```
D:\vue\pinia-counter>npm install pinia
```

အပေါ်က Command Line ကို Run ပြီးရင် Pinia က ကိုယ့် Project ထဲမှာ ရောက်နေပါပြီ။ တကယ်ကော်  
Installed ဖြစ်ရဲ့လားလို့ သေချာချင်ရင်တော့ package.json File ကိုဖွင့်ကြည့်ပါ။ အဲ့ဒီ File မှာ  
dependencies ဆိုတဲ့ Object ထဲမှာ အောက်ပါအတိုင်း pinia ရှိနေရင် အောင်မြင်ပါတယ်။

```
10     "dependencies": {  
11       "core-js": "^3.8.3",  
12       "pinia": "^2.0.24",  
13       "vue": "^3.2.13"  
14     },
```

### Step 2. Register the Pinia

Pinia ကို Install လုပ်ပြီးရင် Vue Counter App Project တစ်ခုလုံးကနေ အသုံးပြုလို့ရအောင်  
အောက်ပါအတိုင်း **src/main.js** မှာ Register လုပ်ပေးရပါတယ်။

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

## အခန်း (၅) – Store Creation & Implementation

### What is a Store?

Pinia ကို Install ကော်၊ Register ကော် လုပ်ပြီးသွားပြီ ဆိုတော့ Data တွေ သိမ်းဆည်းဖို့ Store တစ်ခုဖန်တီး ဖို့ပဲလိုတော့တယ်။ ဒီနေရာမှာ ပြောစရာ တစ်ခုရှိလာတယ်။ Store တစ်ခုကို သတ်မှတ်တော့ မယ်ဆိုရင် ဒီတိုင်း သတ်မှတ်လို့မရဘူး။ သတ်မှတ်ပုံ Store ၂ မျိုးရှိတယ်။ အဲဒါကတော့ **Option Store** နဲ့ **Setup Store** ပဲဖြစ်ပါတယ်။ ကျွန်တော်က တော့ Composition API ကိုပိုသဘောကျတော့ Setup Store ကိုပိုသဘော ကျတယ်။ ဒါပေမယ့် မိတ်ဆွေတို့ ပိုနားလည်အောင် ၂ ခုလုံးကို ပြောပြပေးသွားမယ် ဖြစ်ပါတယ်။

### Option Store

ကဲဒါဆို Option Store ကိုစလေ့လာကြရအောင်။ Option Store မှာ Data တွေကို Manage လုပ်လို့ရတဲ့ Layer Object ၃ ခုရှိပါတယ်။ အဲဒါတွေကတော့ -

- **state**
- **getters**
- **actions**

**state** ဆိုတာ store တစ်ခုရဲ့ main part လို့ဆိုလို့ရတယ်။ Store ဖြစ်လာရတယ်ဆိုတာ state အတွက်ကြောင့်လို့ ဆိုရင်လည်းမမှားဘူး။ state မှာ Data တွေကို Components တွေ Pages တွေကနေ Access လုပ်လို့ရအောင် Data တွေကို Initialize လုပ်ဖို့၊ Store လုပ်ဖို့၊ Define လုပ်ဖို့အသုံးပြုပါတယ်။

**getters** ဆိုတာက state မှာရှိနေတဲ့ data တွေကို Compute လုပ်ပြီး return ပြန်ဖို့အတွက် အသုံးပြုကြပါတယ်။

**actions** ဆိုတာ state မှာရှိနေတဲ့ data တွေကို Modify (add, update, delete) လုပ်ပေးဖို့ အသုံးပြုပါတယ်။

ကဲဒါဆိုရင် Store File တစ်ခုကို ဆောက်ကြမယ်။ src Folder အောက်မှာ stores/CounterStore.js လို့ဆောက်ပေးပါ။ ဆောက်ပြီး သွားပြီဆိုရင် အဲဒီ File ထဲမှာ ရေးရမယ့် Code ကဒီလိုဖြစ်မှာပါ။

```
import { defineStore } from "pinia";

export const useCounterStore = defineStore('counterStore', {

  state: () => {

    return { }

  },

  getters: { },

  actions: { }

})
```

အပေါ်က Code တွေမှာ **Bold** လုပ်ထားတဲ့ အချက် ၅ ချက်ကိုကြည့်ပေးပါ။ ပထမဦးဆုံး Store တစ်ခု သတ်မှတ်ဖို့ရာ pinia Library ကနေ defineStore ဆိုတဲ့ Function ကို Import လုပ်ပေးရပါတယ်။ ပြီးရင် Store နာမည်ကို defineStore ကိုသုံးပြီး သတ်မှတ်ပေးပါ။

defineStore Function က Argument ၂ ခုလက်ခံပါတယ်။ ပထမ တစ်ခုက Store ရဲ့ ID (သို့) နာမည် ဖြစ်ပြီး ကျန်တစ်ခုက state ၊ getters ၊ actions တွေကို Manage လုပ်ရမယ့် Object ဖြစ်ပါတယ်။ ဒုတိယ Argument ဖြစ်တဲ့ Object မှာ **state** ဆိုတဲ့ Function တစ်ခုကို JS ES6 Style နဲ့ရေးထားတာတွေ့ရမယ်။

getters ရယ် actions ရယ်ကိုကျတော့ Object အနေနဲ့ ရေးထားပါတယ် ဒါတွေကို နောက်မှ လေ့လာကြမယ်။

အားလုံးပြီးရင်တော့ counterStore ဆိုပြီး သတ်မှတ်ပေးလိုက်တဲ့ defineStore ကြီးတစ်ခုလုံးကို const useCounterStore ဆိုတဲ့ Variable တစ်ခုထဲကို Assign လုပ်ပြီး export လုပ်ထားရပါတယ်။ အဲဒီလို export လုပ်ထားတာ နဲ့ ဘယ် Components ၊ Pages ကမဆို Access လုပ်လို့ရနေပါပြီ။ တစ်ခုသတ်ထားရမှာက export လုပ်ထားတဲ့ const variable နာမည် ရဲ့အစမှာ use ဆိုပြီး သုံးထားတာ Naming Convention ဖြစ်တယ်လို့ မှတ်ထားပေးပါ။

## 1. State

“state မှာ Data တွေကို Components တွေ Pages တွေကနေ Access လုပ်လို့ရအောင် Data တွေကို Initialize လုပ်ဖို့၊ Store လုပ်ဖို့ ၊ Define လုပ်ဖို့အသုံးပြုပါတယ်”

counterStore ရဲ့ state မှာ Data တွေကို အောက်ပါအတိုင်း သတ်မှတ်ပေးပါ။

```
import { defineStore } from "pinia";

export const useCounterStore = defineStore('counterStore', {

  state: () => {

    return {

      count: 0,

      counterName: 'Pinia Counter'

    }

  },

  getters: { },

  actions: { }

})
```

state Function ထဲမှာ ကိုယ်ကြိုက်သလောက် data တွေကို တည်ဆောက်ပြီး Object တစ်ခုအနေနဲ့ Return ပြန်ထားရပါတယ်။ Code နမူနာမှာ တော့ count ရယ် counterName ရယ် ဆိုပြီး data ၂ခုကို return လုပ်ပြထားပါတယ်။

## Accessing the Store from the Components

ကဲ Store တစ်ခုလည်း တည်ဆောက်ပြီးပြီ၊ အဲဒီ Store ထဲမှာ data ၂ ခုလည်း Declare လုပ်ထားပြီးပြီ ဆိုတော့ App.vue ဆိုတဲ့ Page ကနေသွား Access လုပ်ကြည့်ကြမယ်။ Code က ဒီလိုပါ -

```
<template>

  <div class="container">

    <div>Parent Component</div>

    <h2>{{ counterName }}</h2>

    <h3>Counter - {{ count }}</h3>

    <button>+</button>

    <button>-</button>

  </div>

</template>

<script setup>

  import { useCounterStore } from './stores/CounterStore'

  import { storeToRefs } from 'pinia'

  const counterStore = useCounterStore()

  const { count, counterName } = storeToRefs(counterStore)

</script>

<style>

.container { max-width: 300px; margin: 0 auto; }

button { display: inline-block; margin: 0 5px; padding: 0 10px; font-size: 25px; }

</style>
```



အပေါ်က Code နမူနာကို ရှင်းပြရမယ်ဆိုရင် အရင်ဆုံး အဝါရောင် ကာလာနဲ့ ချယ်ထားတဲ့ အပိုင်း ၃ ပိုင်းကို ကြည့်ပါ။ VueJS လုပ်ဖူးတဲ့ Developer တိုင်းသိမှာပါ။ .vue File မှာ template ၊ script ၊ style ဆိုပြီး ၃ ပိုင်းရှိပါတယ်။ အောက်ဆုံးကနေ စရှင်းပြပေးပါမယ်။

style ဆိုတဲ့ section မှာက Template ထဲက Content တွေကို အလယ်ရောက် အောင်၊ Button တွေကိုနည်းနည်းလှအောင် လုပ်ထားတဲ့ CSS Code အနည်းငယ် ရေးထားပါတယ်။ အဲ့ဒါကို template မှာပြန်သုံးထားပါတယ်။ Pinia အကြောင်းကိုအဓိကထား ရေးသားနေတာကြောင့် CSS Property တစ်ခုချင်းစီတော့ ပြောပြမနေတော့ပါဘူး။

ကဲ script section ကိုကြည့်ကြရအောင်။ <script **setup**> ဒါလေးကိုမြင်ရကတည်းက Vue3 Composition API Style နဲ့ရေးတယ်ဆိုတာ သိထားပါ။ Composition API မှာ ရေးနည်း ၂ နည်းရှိတယ်။ အရမ်းမကွာပါဘူး။

setup ကို Object ပုံစံနဲ့ ရေးတဲ့ပုံစံရယ်၊ အခုလို script tag ထဲမှာ ထည့်ရေးတာရယ်ဆို ပြီး ၂ မျိုးရှိတယ်။ setup ကို Object ပုံစံနဲ့ ရေးတဲ့ style မှာက ကို declare လုပ်ခဲ့တဲ့ Variables တွေ Functions တွေကို Template မှာပြန်သုံးလို့ရအောင် return ပြန်လုပ်ပေးရတယ်။ setup ကိုအခုလို script tag ထဲမှာ ထည့်ရေးတဲ့ style မှာက return ပြန်ပြန်စရာမလိုဘူး။ လောလောဆယ် ဒီလိုပဲ မှတ်ထားပေးပါ။

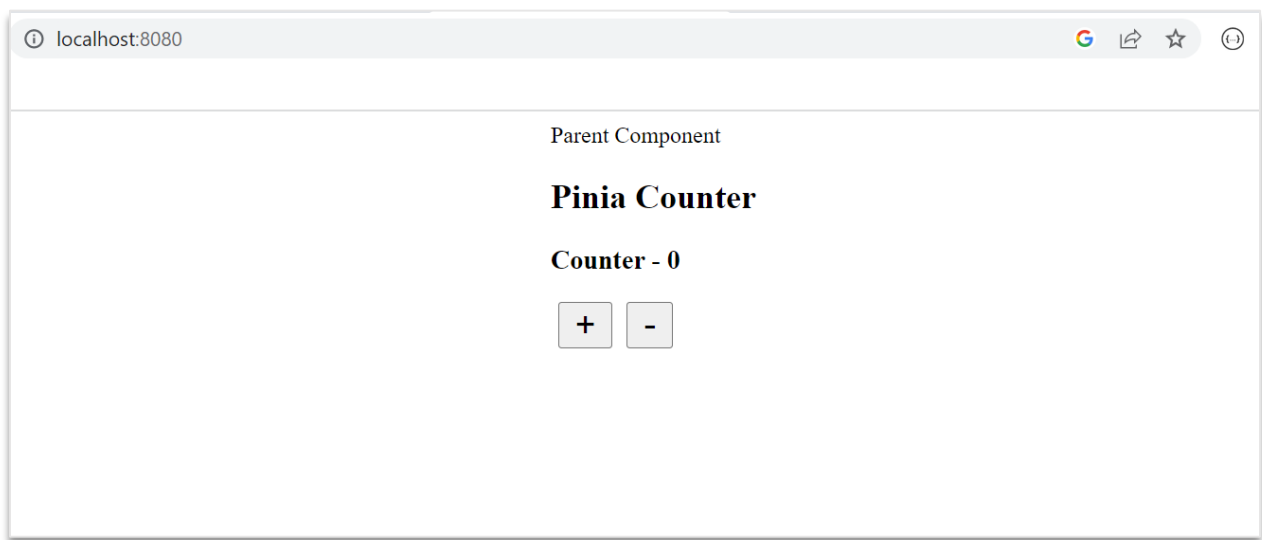
CounterStore.js File ထဲက export လုပ်ထားခဲ့တဲ့ useCounterStore ကိုအသုံးပြုချင်လို့ အဲ့ဒီ CounterStore.js File ကို import လုပ်ပေးထားတာပါ။ pinia Library ထဲက storeToRefs ဆိုတဲ့ Hook Function ကိုလည်း import လုပ်ထားတယ်။

const counterStore ဆိုတဲ့ Variable ထဲကို useCounterStore ကို Assign ချလိုက်ပါတယ်။ ဒါဆိုရင် App.vue ဆိုတဲ့ Page မှာ CounterStore.js ထဲက counterStore ကို Access လုပ်လို့ရနေပါပြီ။ သိထားရမှာက counterStore ထဲမှာ count: 0 ရယ် counterName: “Pinia Counter” ရယ်ဆိုပြီး ref data ၂ ခုရှိတယ်။

အဲ့ဒီ Data ၂ ခုကို App.vue ရဲ့ template မှာ {{ counterStore.count }}၊ {{ counterStore.counterName }} ဆို ပြီး အသုံးပြုလို့ရနေပေမဲ့ မသုံးထားပါဘူး။ {{ count }} ၊ {{ counterName }} ဒီတိုင်းပဲ သုံးချင်လို့ pinia Library က Support ပေးတဲ့ storeToRefs Hook Function ကို သုံးပြီး Destructure လုပ်ထားပါတယ်။ Destructure လုပ်တယ်ဆိုတာ ဒီလိုလေးပါ -

```
const { count, counterName } = storeToRefs(counterStore)
```

အဓိပ္ပါယ်က ဘာလဲဆိုတော့ store data ကနေ ref အဖြစ် ပြောင်းပေးတာပေါ့။ ဒါပေမယ့် ပြောင်းပေးတဲ့ နေရာမှာ state နဲ့ getters တွေကိုပဲ ပြောင်းလို့ရမှာပါ။ နောက်ပိုင်းမှာ ပြောပြမယ့် actions တွေကိုတော့ counterStore ကနေပဲတိုက်ရိုက်ယူသုံးရမှာပါ။ ပြီးရင် template section မှာ +, - နဲ့ button ၂ ခုကိုတွေ့ရမှာပါ။ လုပ်ချင်တာက အဲ့ဒီ Button ၂ခုကို နှိပ်ရင် count တန်ဖိုးကို အတိုးအလျှော့လုပ်ချင်တာပါ။ အဲ့ဒီ Action ကိုတော့ actions Section ရောက်မှ ပြောပြပါမယ်။ Result ကို Browser မှာသွားကြည့်ရင် ဒီလိုလေးမြင်ရပါမယ်။



## 2. getters

“**getters** ဆိုတာက state မှာရှိနေတဲ့ data တွေကို Compute လုပ်ပြီး return ပြန်ဖို့အတွက် အသုံးပြုကြပါတယ်”

App.vue Page မှာ Counter ဆိုပီး {{ count }} ကို Display လုပ်ထားတယ်။ ဆိုကြပါစို့ ကျွန်တော်တို့ အခု လိုချင်တာက Double Count ကိုလိုချင်တယ်။ ဒါဆိုရင် Code ကဒီလိုဖြစ်ပါမယ် -

```
import { defineStore } from "pinia";

export const useCounterStore = defineStore('counterStore', {

  state: () => {

    return {

      count: 0,

      counterName: 'Pinia Counter'

    }

  },

  getters: {

    doubleCount: (state) => state.count * 2

  },

  actions: { }

})
```

အပေါ်က Code တွေအားလုံးကိုတော့ရှင်းပြ စရာမလိုတော့ဘူးထင်တယ်။ ဘာလို့ဆိုတော့ getters မှအပ ရှင်းပြပြီးသားမလိုပါ။ နောက်လည်း ဒီလိုပဲ ရှင်းပြပြီးသား Code နဲ့ မရှင်းပြရသေးတဲ့ Code ဆိုပြီး ၂ မျိုးနဲ့ လာနေ ဦးမှာပါ။

getters ဆိုတဲ့ Object ထဲမှာ doubleCount ဆိုပြီး Function Name တစ်ခုမှာ JS ES6 Anonymous Function တစ်ခုကို Assign ချလိုက်ပါတယ်။ လုပ်ချင်တာက doubleCount ဆိုတဲ့ Function ထဲကို state ထဲမှာ ရှိနေတဲ့ count တန်ဖိုးရဲ့ ၂ ဆကို Return လုပ်ချင်တာပါ။

ဒါဆို state မှာရှိနေတဲ့ count data ကို calculate လုပ်ရမှာမလို့ state ကို Anonymous Function ရဲ့ Argument အနေနဲ့ pass လုပ်ပေးရင် အဲ့ဒီ Function ထဲမှာ state ကို အသုံးပြုလို့ရပါပြီ။ `state.count * 2` ဆိုတာက state မှာရှိတဲ့ count တန်ဖိုးကို 2 နဲ့မြှောက်လိုက် တာဖြစ်ပါတယ်။ တကယ်တော့ doubleCount ဆိုတဲ့ getter ကို App.vue ကနေ Access လုပ်လို့ရနေပါပြီ၊ ဒါပေမဲ့ မလုပ်သေးပါဘူး၊ actions တွေလုပ်ပြီးမှ တစ်ပါတည်းလုပ်တော့မယ်။

### 3. actions

“**actions** ဆိုတာ state မှာရှိနေတဲ့ data တွေကို Modify (add, update, delete) လုပ်ပေးဖို့ အသုံးပြုပါတယ်”

App.vue မှာရှိနေတဲ့ အပေါင်း Button နှိပ်ရင် Counter တန်ဖိုး ၁ စီတိုးစေချင်တယ်၊ အနုတ် Button နှိပ်လိုက်ရင် Counter တန်ဖိုး ၁ စီလျှော့စေချင်တယ်။ ဒါဆို Code ကဒီလိုဖြစ်မှာပါ -

```
import { defineStore } from "pinia";

export const useCounterStore = defineStore('counterStore', {

  state: () => {

    return {

      count: 0,

      counterName: 'Pinia Counter'

    }

  },

  actions: {

    increment() {

      this.count ++

    },

    decrement() {

      if(this.count > 0) {

        this.count --

      }

    }

  }

})
```

Code တွေက နည်းနည်းများနေတော့ မဆန့်တော့လို့ getters Section ကိုခန့်ဖျက်ထားပါတယ်။ VS Code မှာတော့ သွားမဖျက်နဲ့ ဦး။ actions ဆိုတဲ့ Object ထဲမှာ increment နဲ့ decrement ဆိုပြီး Function ၂ ခုကိုတွေ့ရမှာပါ။ Button တွေကို Click လုပ်လိုက်တာနဲ့ state ထဲမှာရှိနေတဲ့ count data ကို increment၊ decrement လုပ်ပေးရမှာမို့ state ကို this Keyword နဲ့ Access လုပ်ပြီး ++, -- နဲ့ ၁ စီ အတိုး အလျှော့ လုပ်လိုက်တာပါ။

ကဲ Store အပိုင်းကို ပြီးသွားပြီဆိုတော့ App.vue ကိုသွားပြီး Store ထဲက getters တို့ actions တွေကို သွား Access လုပ်ကြရအောင်ပါ။ Code ကဒီလိုဖြစ်မှာပါ -

```
<template>

  <div class="container">

    <div>Parent Component</div>

    <h2>{{ counterName }}</h2>

    <h3>Counter - {{ count }}</h3>

    <h3>Double Counter - {{ doubleCount }}</h3>

    <button @click="counterStore.increment">+</button>

    <button @click="counterStore.decrement">-</button>

  </div>

</template>

<script setup>

  import { useCounterStore } from './stores/CounterStore'

  import { storeToRefs } from 'pinia'

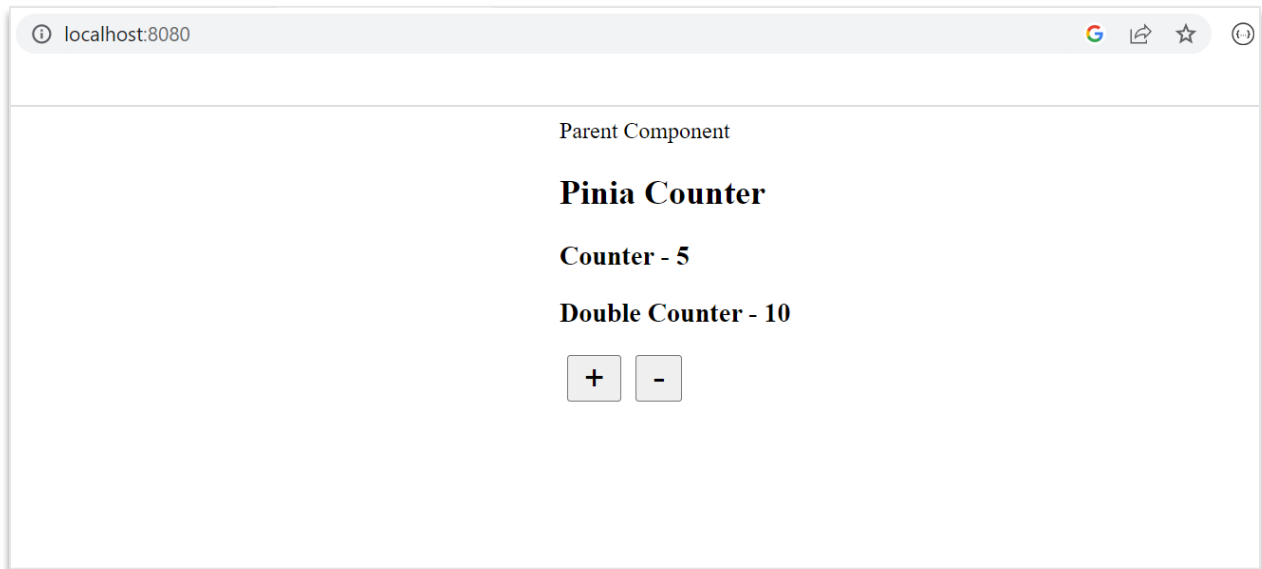
  const counterStore = useCounterStore()

  const { count, counterName, doubleCount } = storeToRefs(counterStore)

</script>
```

**Bold** လုပ်ထားတဲ့ Code တွေကိုပဲကြည့်ပေးပါ။ template ကိုမပြောမီ script section ကိုအရင် ပြောလိုက်ပါမယ်။ ကျွန်တော် အပေါ်မှာလည်း ပြောခဲ့တယ်။ အဲဒါက ဘာလဲဆိုတော့ Store ထဲက state နဲ့ getters တွေကိုပဲ Destructure လုပ်လို့ရပါတယ်။ actions တွေကို ကျတော့ counterStore ကနေပဲ တိုက်ရိုက် ယူသုံးရပါမယ်။

ဒါကြောင့် doubleCount ကို Destructure လုပ်ပြီး template မှာ အသုံးပြုထားပါတယ်။ Store ရဲ့ increment နဲ့ decrement ကိုတော့ Button လေးတွေနဲ့ရင် အလုပ်လုပ်အောင် counterStore ကနေ dot နဲ့ တိုက်ရိုက်ခေါ်လိုက်ပါတယ်။ ပြီးသွားရင် Browser ကို Refresh လုပ်ပြီး Button လေးတွေကို Click လုပ်ကြည့်ပါ။ အဆင်ပြေသွားမှာပါ။ Result ကတော့ ဒီလိုဖြစ်လာ ပါမယ် -



## Accessing a Store from Multiple Components

ကျွန်တော်တို့ Counter App က တော်တော်ကို အဆင်ပြေနေပါပြီ။ နောက်တစ်ခုဘာ လုပ်ချင်သေးလဲဆိုတော့၊ ဆိုပါစို့ နောက်ထပ် Child Component ၂ ခုကို တည်ဆောက်မယ်။ အဲဒီ Component လေးတွေထဲမှာ counterStore ကို import လုပ်ပြီး သူ့ထဲက count နဲ့ doubleCount data တွေကို Display ပြမယ်။ ပြီးရင် actions ထဲက Increment၊ Decrement ဆိုတဲ့ Function တွေကိုလည်း Buttons လေးတွေမှာယူသုံးမယ်။ ကဲဒါဆို ComponentB.vue ရယ် ComponentC.vue ရယ်ဆိုပြီး src/components Folder ထဲမှာ ဆောက်ပေးပါ။ ပြီးတော့ အဲဒီ တွေထဲမှာ ကျွန်တော် ပြောထားသလို code တွေရေးပေးပါ။ ဒီလိုပါ -

```
<template>

  <div class="container">

    <div>Component B</div>

    <h3>Counter - {{ count }}</h3>

    <h3>Double Counter - {{ doubleCount }}</h3>

    <button @click="counterStore.increment">+</button>

    <button @click="counterStore.decrement">-</button>

  </div>

</template>

<script setup>

  import { useCounterStore } from '../stores/CounterStore'

  import { storeToRefs } from 'pinia'

  const counterStore = useCounterStore()

  const { count, doubleCount } = storeToRefs(counterStore)

</script>
```



အပေါ်က Code တွေကတော့ ComponentB.vue မှာရေးရမယ့် Code တွေပါ။ ComponentC.vue မှာလည်း ဒီတိုင်း ပဲရေးပေးရမှာဆိုတော့ မဖော်ပြနေတော့ပါဘူး။ ခေါင်းစဉ်မှာ Component B အစား Component C လို့ပြောင်းရေးရင် ရပါပြီ။

အဲ့ဒါတွေ အကုန်လုံး ရေးပြီးတော့ App.vue ဆိုတဲ့ Parent Component ကိုသွားပြီး Child Component ၂ ခုကို import လုပ်ပြီး အောက်ပါအတိုင်း အသုံးပြု ပေးပါ။

```
<template>

  <div class="container">

    ...

    <hr>

    <ComponentB />

    <hr>

    <ComponentC />

  </div>

</template>

<script setup>

  import { useCounterStore } from './stores/CounterStore'

  import { storeToRefs } from 'pinia'

  import ComponentB from './components/ComponentB.vue'

  import ComponentC from './components/ComponentC.vue'

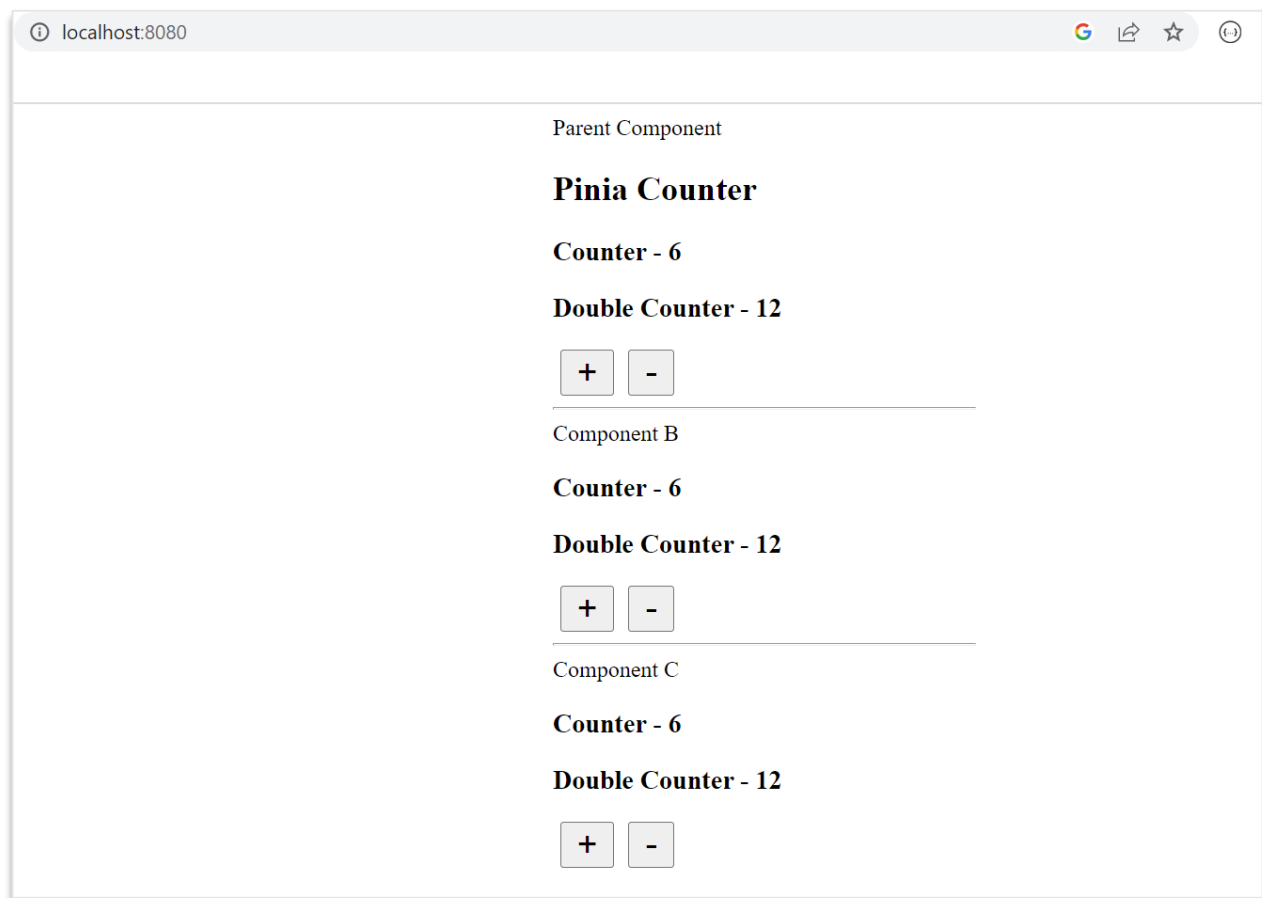
  const counterStore = useCounterStore()

  const { count, counterName, doubleCount } = storeToRefs(counterStore)

</script>
```

Code တွေများနေလို့ ရှင်းပြပြီးသား code တွေကို ခန့်ဖျက်ထားပါတယ်။ VS Code မှာတော့သွား မဖျက်လိုက်ပါနဲ့။

ဘာလို့ Multiple Components တွေထဲမှာ counterStore ကို import လုပ်ပြီး increment၊ decrement လုပ်နေသလဲဆိုတော့ ဘယ် Component မှာပဲ increment၊ decrement လုပ်ပါစေ တစ်ချိန်တည်း တစ်ပြိုင်တည်းမှာ Application တစ်ခုလုံးရဲ့ ဘယ်နေရာမှာ မဆို store Data တွေကို ယူသုံးထားတဲ့ နေရာမှန်သမျှမှာ Effect ဖြစ်ပါတယ်ဆိုတဲ့ ဟာကိုသက်သေ ပြချင်လို့ဖြစ်ပါတယ်။ Final Result က ဒီလိုဖြစ်ပါမယ် -



Source Code ကိုဒီကနေလည်း ရယူလို့ရပါတယ်။

<https://github.com/yemyintsoe/pinia-vue3-counter-app>

## Setup Store

ကဲ ဒါဆိုရင် Option Store နဲ့ Store တစ်ခုကိုဘယ်လို တည်ဆောက်မလဲ ဆိုတာလေ့လာ ကြပြီးပြီ ဆိုတော့ နောက်ထပ် ကျန်နေသေးတဲ့ **Setup Store** ကိုထပ်လေ့လာကြပါမယ်။ Option Store ကိုနားလည်ပြီးသွားရင် Setup Store ကိုလည်း တန်းရေးတတ်နေပါပြီ အခုကျန်တော် ပြောမယ့် အချက်လေးတွေကို သိထားရင်ပေါ့။ Setup Store အတွက်နောက်ထပ် အကြောင်းအရာ အသစ်နဲ့ မပြောပါဘူး Option Store နဲ့ရေးထားတဲ့ Counter app အတွက်ပဲ ပြန်ရေးကြမှာပါ။ ဒါမှလည်း Option Store နဲ့ Setup Store တို့ရဲ့ ကွာခြားမှုကို ကြည့်လို့ရမှာပေါ့။ Setup Store အတွက် Code ကဒီလိုပါ -

```
import { defineStore } from "pinia";

import { computed, ref } from "vue";

export const useCounterStore = defineStore('counterStore', () => {

  // state

  const count = ref(0)

  const counterName = ref('Pinia Counter')


  // getters

  const doubleCount = computed(( ) => count.value * 2)


  // actions

  function increment() {

    count.value ++

  }

  function decrement() {

    if(count.value > 0) { count.value -- }

  }

  return {count, counterName, doubleCount, increment, decrement}
```

အပေါ်က Setup Store Code ကိုကြည့်ရင် Option Store နဲ့ သိပ်ကွာတာ မတွေ့ရပါဘူး။ Store တစ်ခုဖန်တီးဖို့ defineStore ကို import လုပ်တာတွေ၊ defineStore နဲ့ Store ကို သတ်မှတ်ပေးတာတွေလည်း တူပါတယ်။ ဒါပေမယ့် Setup Store ဖန်တီးတဲ့ အချိန်မှာတော့ defineStore ရဲ့ ဒုတိယ Argument က Object မဟုတ်တော့ပါဘူး Function ဖြစ်သွားပါပြီ။ JS ES6 Function ကိုသုံးထားတာ တွေ့ရမှာပါ။

အဓိက ပြောင်းလဲသွားတာ က state ၊ getters နဲ့ actions တွေပဲဖြစ်ပါတယ်။ အပေါ်က Code တွေကို ကြည့်လိုက်ရင် state ၊ getters နဲ့ actions တွေကို တွေ့ရတော့မှာ မဟုတ်ပါဘူး။ ဒါတွေက ဘာဘဲလို ဖြစ်သွားတာလဲ ဆိုတော့ -

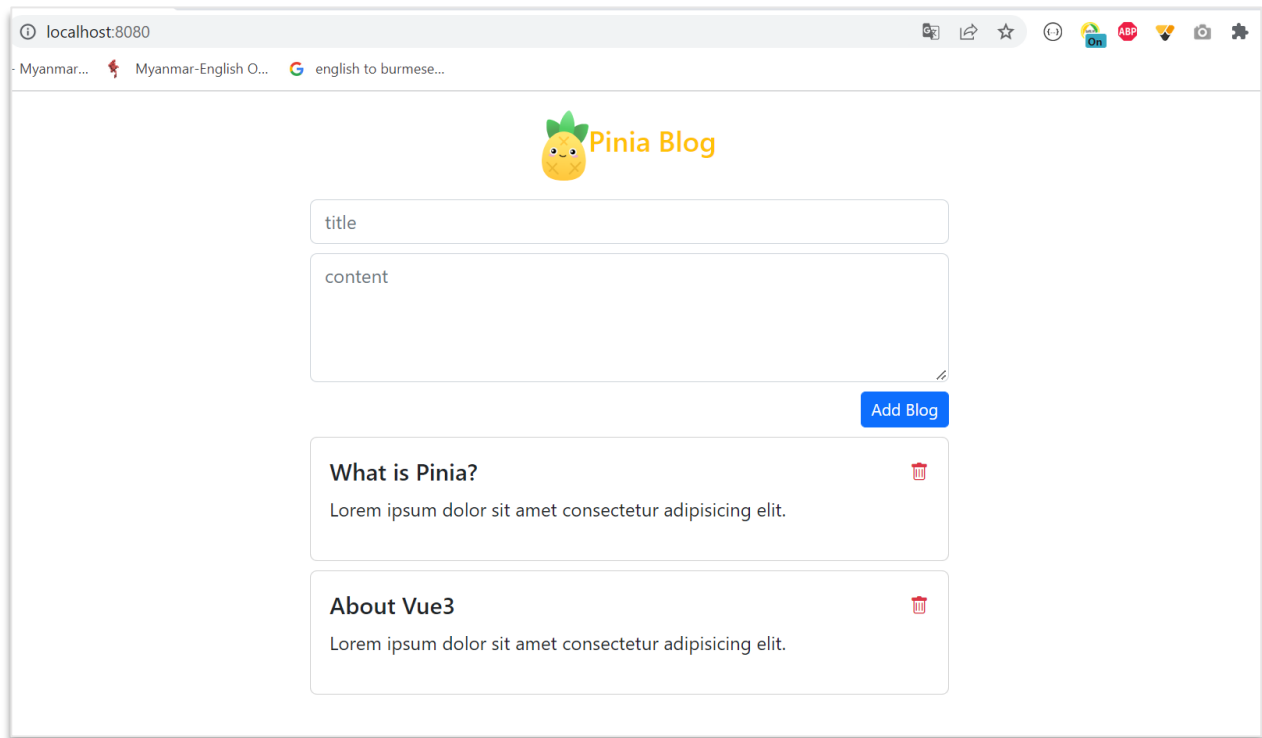
Setup Store ထဲမှာဆိုရင်

- **state** အစား **ref()**
- **getters** အစား **computed()**
- **actions** အစား **function()**

ဒီလိုတွေ ဖြစ်သွားကြပါတယ်။ ပြောရရင် Code တွေက ပိုသန့်သွားပြီး ရှင်းသွားပါတယ်။ နားလည်ဖို့ လည်း အရမ်း လွယ်တယ်။ Vue File ဘက်မှာ Composition API ရေးနည်းနဲ့ လုံးဝ တစ်ထပ်တည်း တူသွားပါတယ်။

state နဲ့ getters အစား ref() နဲ့ computed() ကိုသုံးရလို့ အပေါ်မှာတော့ import လုပ်ထားရပါမယ်။ actions အစား function() ကိုသုံးရတဲ့ နေရာမှာတော့ ဘာမှ import လုပ်စရာမလို အပ်ပါဘူး။ နောက်ထပ် သိထားရမှာ က Option Store မှာတုန်းက state ထဲမှာရှိနေတဲ့ count ကို getters နဲ့ actions ကနေ Access လုပ်ဖို့ state.count (သို့) this.count ဆိုပြီး ခေါ်ရပါတယ်။ ဒါပေမယ့် Setup Store မှာတော့ ဒီတိုင်း Composition API မှာခေါ်တဲ့ အတိုင်း count.value ဆိုပြီးခေါ်ရပါတယ်။ နောက်ဆုံး အဆင့်အနေနဲ့ သတိထားရမယ့် အချက်တစ်ခု ရှိပါတယ်။ အဲဒါက ဘာလဲဆိုတော့ ref တွေ computed တွေ function တွေကို Object တစ်ခု အနေနဲ့ return လုပ်ထားပေးရပါမယ်။ return မလုပ်ထားခဲ့ဘူးဆိုရင် Components တွေကနေ ခေါ်သုံးလို့ရမှာ မဟုတ်ပါဘူး။ အပေါ်က အတိုင်း Code တွေကို ရေးပြီးသွားရင်တော့ Vue File ဘက်မှာ ဘာမှ သွားပြောင်းစရာ မလိုတော့ပါဘူး။ Browser ကို Refresh လုပ်ပြီး ပြန်စမ်းကြည့်လို့ရပါပြီ။ အားလုံးအဆင်ပြေသွားမှာပါ။

## အခန်း (၆) – Pinia Blog App Using Vue3, Pinia & Bootstrap5



ကဲ Pinia ကိုတော်တော်လည်း နားလည်သွား ပြီဆိုတော့ ဒီထက်ပိုနားလည်အောင် နဲ့ Real World Project တွေရေးနိုင်အောင် နောက်ထပ် Blog App လေးတစ်ခုကို ဖန်တီးပေးပါမယ်။ ဒါပေမယ့် တစ်ခုတော့ ပြောချင်ပါတယ်၊ အဲ့ဒါကဘာလဲ ဆိုတော့ Counter app လေးဖန်တီး တုန်းကလို တော့ Code တစ်ကြောင်းချင်းစီကို အသေးစိတ် ပြောပြတော့မှာ မဟုတ်ပါဘူး။ ဒီ Blog app မှာပါမဲ့ Feature တွေကတော့ CRD ပဲဖြစ်ပါတယ်။ C ဆိုတာ Create ပါ၊ Blog အသစ်တွေကို ထည့်လို့ရတဲ့ Feature ပါ။ R ဆိုတာ Read or Retrieve ပါ၊ Blog တွေကို ဆွဲထုတ်ပြီး Display ပြရမဲ့ Feature ပါ။ D ဆိုတာကတော့ Delete or Destroy ပါ၊ Blog တွေကို ဖျက်လို့ရတဲ့ Feature ဖြစ်ပါတယ်။

အခု ရေးမယ့် Pinia Blog App လေးကို လိုက်လုပ်နိုင်ဖို့ မိတ်ဆွေတို့က Vue3 Composition API ကို အသင့်သင့် ရထားမှ ဖြစ်ပါမယ်။ ဒီ app လေးကိုသာ ရအောင်လိုက်ရေး နိုင်ရင် မိတ်ဆွေက Vue3 Composition API နဲ့ Pinia ကိုတော်တော် သုံးတတ်နေပြီ ဆိုတဲ့သဘောပါ။ နောက်တစ်ခု ထပ်သိလာ

မှတ် Vue app မှာ Bootstrap ကိုဘယ်လို Install လုပ်ပြီး ဘယ်လိုသုံးမလဲ ဆိုတဲ့ Knowledge ကိုပါရသွားမှာပါ။ ကဲဒါဆိုရင် စလိုက်ကြရအောင်။

## Create Vue App

Vue app တစ်ခုကို နာမည်တစ်ခု ပေးပြီး ဖန်တီးလိုက်ပါ။ ဘယ်လို ဖန်တီးရမလဲဆိုတာတော့ အပေါ်မှာ ပြောပြီးသားမလို့ မပြောတော့ပါဘူး။ ကျွန်တော်ကတော့ pinia-blog-app လို့နာမည်ပေးထားပါတယ်။

## Install Pinia & Register it to Vue app

အခုနားက ဖန်တီးထားတဲ့ vue blog app ထဲမှာ Pinia ကို Install လုပ်ပြီး Register ပြုလုပ်ပေးပါ။

## Install Bootstrap5 & use it

ဒီ Blog app ရဲ့ Design ပိုင်းအတွက် Bootstrap ကိုသုံးမှာမလို့ Bootstrap ကို Install လုပ်ဖို့ အောက်က Command Line ကို Run ပေးပါ။

Version အတိအကျမရေးပဲ၊ npm install bootstrap ဒီတိုင်း Run လည်းရပါတယ်။

```
D:\vue\pinia-blog-app>npm install bootstrap@5.2.2
```

Bootstrap Icon Library ကိုအသုံးပြုမှာမလို့ အောက်ပါ Command Line ကို Run ပြီး Install လုပ်ပေးပါ။

```
D:\vue\pinia-blog-app>npm i bootstrap-icons
```

Bootstrap ကော် Bootstrap Icon ကော် Installed လုပ်ပြီးသွားပြီဆိုတော့ သူတို့ကို Register (or) Use ဖို့ပဲလိုတော့တယ်။ တကယ် Installed ဖြစ်လာ သေချာသိချင်ရင်တော့ package.json ရဲ့ dependencies Object အောက်မှာသွား ကြည့်လို့ရပါတယ်။

Vue app မှာ Bootstrap ကို Installed လုပ်ပြီးရင် Project တစ်ခုလုံးမှာ အသုံးပြုလို့ရအောင် အောက်ပါ အတိုင်း main.js File မှာ import လုပ်ပေးရပါတယ်။

```

JS main.js M X
src > JS main.js
1  import { createApp } from 'vue'
2  import App from './App.vue'
3  import { createPinia } from 'pinia'
4
5  import "bootstrap/dist/css/bootstrap.min.css"
6  import "bootstrap-icons/font/bootstrap-icons.css"
7
8  createApp(App)
9  .use(createPinia())
10 .mount('#app')
11
12 import "bootstrap/dist/js/bootstrap.bundle.min.js"
13

```

အပေါ်ကပုံကတော့ main.js File ပါ။ bootstrap ကို ၃ နေရာမှာ import လုပ်ထားပါတယ်။ တကယ်တော့ Bootstrap အတွက်ကတော့ ၂ ခုပါ။ bootstrap.min.css နဲ့ bootstrap.bundle.min.js ဆိုတဲ့ ၂ ခုပါ။ နောက်ထပ် bootstrap-icons.css ဆိုတာက Bootstrap Icon အတွက်ဖြစ်ပါတယ်။ မိတ်ဆွေတို့ Code လည်းအပေါ်က တိုင်းဖြစ်လားကြည့်ပါ မတူရင် မှားနိုင်ပါတယ်။

Vue ၊ Pinia ၊ Bootstrap ၊ Bootstrap Icon ဒါတွေ အားလုံးကို Installed လုပ်ပြီး Configuration လည်းလုပ်ပြီး သွားပြီဆိုတော့ ကျွန်တော်တို့ရေးချင်တဲ့ Pinia Blog App လေးစရေးလို့ရနေပါပြီ။

## Creating a Store for Blog app

ပထမဆုံးအနေနဲ့ blog app လေးအတွက် Store တစ်ခုဖန်တီးကြမယ်။ src/stores/BlogStore.js ဆိုပြီး File တစ် File ဆောက်လိုက်ပါ။ အဲ့ဒီ File ထဲမှာ Code တွေက အောက်ပါအတိုင်း ဖြစ်ပါမယ်။

```

JS BlogStore.js U X
src > stores > JS BlogStore.js > [x] useBlogStore
1  import { defineStore } from "pinia";
2  import { ref } from "vue";
3  export const useBlogStore = defineStore('blogStore', () => {
4      const blogs = ref([
5          {
6              id: 1,
7              title: "What is Pinia?",
8              content: "Lorem ipsum dolor sit amet consectetur adipisicing elit."
9          },
10         {
11             id: 2,
12             title: "About Vue3",
13             content: "Lorem ipsum dolor sit amet consectetur adipisicing elit."
14         }
15     ])
16     function addBlog(formData) {
17         blogs.value.push(formData)
18     }
19     function deleteBlog(id) {
20         blogs.value = blogs.value.filter(blog => blog.id !== id)
21     }
22     return { blogs, addBlog, deleteBlog }
23 }

```

Blog app လေးအတွက် ပြည့်စုံပြီးသား Store File ဖြစ်နေတော့ Code တွေကတော့ နည်းနည်းများပါလိမ့်မယ်။ ဒါပေမယ့် သေချာလေး ကြည့်ပေးပါ။ Counter app နဲ့ အရမ်းကြီး ဘာမှ မကွာနေပါဘူး။ ဒီ Store ကိုသေချာလေး ကြည့်လိုက်ရင် ref တစ်ခုနဲ့ function နှစ်ခုကို တွေ့ရမှာပါ။ ref က blog လေးတွေကို စုပေးထားတဲ့ array ဖြစ်ပါတယ်။ အဲ့ဒီ array ထဲမှာ blog Object လေးနှစ်ခုကို ကြိုထည့်ထားပါတယ်။ addBlog ဆိုတဲ့ function လေးကတော့ Form ကနေ အသစ်ထည့်လိုက်တဲ့ form-data တွေကို Argument အနေနဲ့ လက်ခံပြီး blog ဆိုတဲ့ ref ထဲကို push function ကိုသုံးပြီး ထပ်ထည့်တဲ့ အလုပ်ကို လုပ်ပါတယ်။ deleteBlog function ကတော့ blogs လေးတွေကို Display ပြထားတဲ့ List ထဲက တစ်ခုခုကို ဖျက်လိုက်ရင် အလုပ်လုပ်ပေးမယ့် function ပါ။ ဒီ function ကလည်း List ကနေ Click လုပ်လိုက်တဲ့ id ကို Argument အနေနဲ့ လက်ခံပြီး အပေါ်က blogs ဆိုတဲ့ ref ကနေ filter လုပ်လိုက်



တာပါ။ နောက်ဆုံးအနေနဲ့ ref တွေ function တွေကို Component တွေကနေ အသုံးပြုလို့ရအောင် return လုပ်ထားရပါမယ်။

### Creating Components & Implementation

အပေါ်မှာ Store တစ်ခုကို ဖန်တီးပြီး blog app လေးအတွက် လိုအပ်တဲ့ Store နဲ့ သက်ဆိုင်တဲ့ Code တွေကို ရေးခဲ့ကြပါတယ်။ Form ကနေ အသစ်ထည့်လိုက်တဲ့ Form Data တွေ ဘာညာတွေသာ ပြောခဲ့တာ မျက်စိထဲမှာ မမြင်ယောင်ဘူးဖြစ်နေတတ်ပါတယ်။ တကယ်တော့ Form တွေကို အရင်ဆောက်ပြီး add new button တွေလုပ်ပြီးမှ store ထဲမှာ action တွေရေးသင့်တာ၊ ဒါပေမယ့် ပုံတွေ အရမ်းများသွား မှာစိုးလို့ File တစ်ခုအတွက် သူတို့နဲ့ သက်ဆိုင်တဲ့ Code အပြည့်စုံ တစ်ပါတည်း ထည့်လိုက်တာပါ။

ကဲဒါဆိုရင် Component File ၃ ခုဖန်တီးကြရအောင်။ တကယ်တော့ App.vue ဆိုတဲ့ File ကရှိနေပြီးသား ပဲဆိုတော့ components Folder ထဲမှာပဲ BlogCreate.vue နဲ့ BlogItem.vue ဆိုပြီး နောက်ထပ် ၂ File ကိုထပ်ဖန်တီးပေးပါ။

App.vue File မှာရေးရမယ့် Code တွေက ဒီလိုပါ။

```

App.vue M X
src > App.vue > Vetur > {} "App.vue" > script
1  <template>
2    <div class="container">
3      <div class="row">
4        <div class="d-flex align-items-center justify-content-center my-3">
5          
6          <h4 class="text-warning text-center">Pinia Blog</h4>
7        </div>
8        <div class="col-md-6 offset-md-3">
9          <BlogCreate />
10         <BlogItem v-for="blog in blogs" :key="blog.id" :blog="blog" />
11       </div>
12     </div>
13   </div>
14 </template>
15 <script setup>
16   import { useBlogStore } from './stores/BlogStore'
17   import { storeToRefs } from 'pinia'
18   import BlogCreate from './components/BlogCreate.vue'
19   import BlogItem from './components/BlogItem.vue'
20
21   const blogStore = useBlogStore()
22   const { blogs } = storeToRefs(blogStore)
23 </script>

```

App.vue File က Code တွေကို ရှင်းပြရရင် template ရယ် script ရယ်ဆိုပြီး အပိုင်း ၂ ပိုင်းနဲ့ ကြည့်ပေးပါ။ template မှာဆိုရင် Bootstrap class တွေကို ခေါ်သုံးထားတာကို သတိပြုပါ။ script မှာ BlogCreate နဲ့ BlogItem ဆိုတဲ့ Child Component ၂ ခုကို import လုပ်ပြီး template ရဲ့ col-md-6 ဆိုတဲ့ div tag အတွင်းမှာ ပြန်ပြီး အသုံးပြုထား တာပါ တွေ့ရမှာပါ။ အဲ့ဒီ Component ၂ ခုကိုလည်း ခန နေရင် ဖန်တီး တော့မှာပါ။ BlogCreate ဆိုတဲ့ Child Component ဆိုတာ blog အသစ်တွေ ဖန်တီးလို့ရမယ့် Form အတွက်ဖြစ်ပါတယ်။ BlogItem ကတော့ blog တွေကို List အနေနဲ့ Display ပေးလုပ်မယ့် Component ဖြစ်ပါတယ်။

BlogStore.js ကနေ store ကိုလည်း သွား import လုပ်ထားပါတယ်။ ပြီးရင် storeToRefs ကိုသုံးပြီး အဲ့ဒီ store ကို Destructure လုပ်လိုက်ပါတယ်။ Destructure လုပ်လို့ရလာတဲ့ store ထဲက blogs ref ကို BlogItem မှာ Loop ပတ်ပြီး Child Component ကနေ Access လုပ်လို့ရအောင် pass လုပ်လိုက်ပါတယ်။ ဒီလောက်ဆိုရင် App.vue ကိုနားလည်မယ်လို့ ထင်ပါတယ်။ Browser မှာတော့ သွား Run လို့ရဦးမှာ

မဟုတ်ပါဘူး။ Error တက်နေဦးမှာပါ။ BlogItem ဆိုတဲ့ Child Component ကို src/components Folder မှာ ဖန်တီးကြပါမယ်။

BlogItem.vue File မှာ ရေးရမယ့် Code တွေက ဒီလိုပါ။



```

1 <template>
2   <div class="card my-2">
3     <div class="card-body">
4       <div class="d-flex justify-content-between">
5         <h5>{{ blog.title }}</h5>
6         <span class="text-danger" role="button" @click="blogStore.deleteBlog(blog.id)">
7           <i class="bi bi-trash"></i>
8         </span>
9       </div>
10      <p>{{ blog.content }}</p>
11    </div>
12  </div>
13 </template>
14
15 <script setup>
16   import { defineProps } from 'vue'
17   import { useBlogStore } from '../stores/BlogStore'
18   defineProps({
19     blog: Object
20   })
21   const blogStore = useBlogStore()
22 </script>

```

BlogItem.vue File ကတော့ ရှင်းပါတယ်။ Parent Component ဖြစ်တဲ့ App.vue ကနေပို့လိုက်တဲ့ data ကို defineProps function နဲ့ ရယူပြီး template မှာ Display လုပ်လိုက်တာပါ။ တကယ်တော့ defineProps က Vue3 Composition API ရဲ့ <script setup> ကိုသုံးလိုက်ရင် import လုပ်စရာတောင်မလိုပါဘူး။ ဒါပေမဲ့ မိတ်ဆွေတို့ဆီမှာ import မလုပ်မိလို့ error တက်သွားရင် package.json မှာ သွားပြီး မပြင်တတ်မှာ စိုးလို့ defineProps ကို import လုပ်လိုက်တာပါ။

နောက်တစ်ခုက store ကို access လုပ်လိုက်ပါတယ်။ အမှိုက်ပုံး Icon လေးကို Click တဲ့အချိန်မှာ blog ကိုဖျက်တဲ့ အလုပ်လုပ်စေချင်လို့ store ထဲမှာ ရေးထားခဲ့တဲ့ deleteBlog function ကို ယူသုံးလိုက်ပါတယ်။ deleteBlog function က Argument တစ်ခုလိုနေလို့ blog.id ဆိုပြီး blog id ကို pass လုပ်ပေးလိုက်ပါတယ်။ အခုထိလည်း Browser မှာ တော့ Error တက်နေဦးမှာပါ။ ဘာလို့ဆိုတော့ App.vue မှာသုံးလိုက်တဲ့ Child Component ၂ခုမှာ တစ်ခုကိုပဲ တည်ဆောက်လို့ပြီးနေသေးလို့ပါ။

ကဲ ဒါဆိုရင် နောက်ဆုံးကျန်နေတဲ့ BlogCreate.vue ကို src/components/ Folder မှာသွားဖန်တီးပေးပါ။  
အဲဒီ File မှာ ရေးရမဲ့ Code ကဒီလိုဖြစ်မှာပါ။

```

▼ BlogCreate.vue U X
src > components > ▼ BlogCreate.vue > Vetur > {} "BlogCreate.vue" > template
1  <template>
2    <form @submit.prevent="addNewBlog">
3      <div class="mb-2">
4        <input type="text" v-model="newTitle"
5          class="form-control" placeholder="title" required>
6      </div>
7      <div class="mb-2">
8        <textarea rows="4" v-model="newContent"
9          class="form-control" placeholder="content" required></textarea>
10     </div>
11     <div class="clearfix">
12       <button class="btn btn-primary btn-sm float-end">Add Blog</button>
13     </div>
14   </form>
15 </template>

17 <script setup>
18   import { ref } from 'vue';
19   import { useBlogStore } from '../stores/BlogStore'
20   const blogStore = useBlogStore()
21   let newTitle = ref('')
22   let newContent = ref('')
23   function addNewBlog(){
24     const newBlog = {
25       id: Math.floor(Math.random() * 1000),
26       title: newTitle.value,
27       content: newContent.value
28     }
29     blogStore.addBlog(newBlog)
30     // clear the form
31     newTitle.value = ''
32     newContent.value = ''
33   }
34 </script>

```

BlogCreate.vue File မှာတော့ Code တွေနည်းနည်းများနေတော့ template ရယ် script ရယ်ဆိုပြီး ပုံ  
ပုံနဲ့ခွဲထားရပါတယ်။ အပေါ်က Code တွေကို ကြည့်လိုက်ရင် template အပိုင်းမှာတော့ Bootstrap

Class တွေနဲ့ Blog အသစ်တွေ ထည့်လို့ရမယ့် Form တစ်ခုကို ဆောက်ထားပါတယ်။ script အပိုင်းမှာတော့ template က Form မှာ Data တွေထည့်ပြီး Submit လုပ်လိုက်တဲ့ အခါမှာ addBlog ဆိုတဲ့ function ကို ယူသုံးချင်လို့ BlogStore.js ထဲက Store ကို import လုပ်ထားပါတယ်။

Form Input တွေရဲ့ v-model အတွက် variable တွေတည်ဆောက်ဖို့ ref ကို import လုပ်ထားပါတယ်။ Form ကို Submit လုပ်တဲ့ အခါမှာ Fire လုပ်ဖို့ addNewBlog ဆိုတဲ့ function တစ်ခုကို ဆောက်လိုက်ပါတယ်။ အဲဒီ function ထဲမှာ newBlog ဆိုတဲ့ နာမည်နဲ့ store ထဲက blogs ရဲ့ Object တွေနဲ့ ပုံစံတူ Object တစ်ခုကို ဆောက်လိုက်ပါတယ်။ ပြီးရင် store ထဲက addBlog ဆိုတဲ့ function ကိုခေါ်သုံးလိုက်ပါတယ်။ အဲဒီ function ကိုသုံးဖို့ Argument တစ်ခုကို pass ပေးရမှာမလို့ အစောနားက ဖန်တီးလိုက်တဲ့ newBlog ဆိုတဲ့ Object ကို pass လုပ်လိုက်ပါတယ်။ နောက်ဆုံးမှာတော့ Form ကို Clear လုပ်လိုက်တာပါ။ ဒီစာအုပ်လေးက Pinia Store အကြောင်းကိုပဲ အဓိကထားပြောပေး နေတာကြောင့် VueJS3 Composition API Code တစ်ခုချင်းစီရဲ့ အလုပ်လုပ်ပုံကို သေချာ မပြောနေတာ သတိထားကြမယ် လို့ ထင်ပါတယ်။ အပေါ်က Code အတိုင်းရေးပြီးရင်တော့ Browser ကို Refresh လုပ်ပြီး စမ်းကြည့်လို့ရပါပြီ။ အဆင်ပြေ သွားမှာပါ။

Source Code ကိုဒီကနေလည်း ရယူလို့ရပါတယ်။

<https://github.com/yemyintsoe/pinia-blog-app>

## နိဂုံးချုပ်

ကဲဒါဆိုရင် **Pinia** ကဘယ်လောက်ထိ အသုံးဝင်တယ်၊ ဘယ်လို အသုံးပြု ရမယ်ကို တော်တော်လေး နားလည်ကြပြီလို့ယုံကြည်ပါတယ်။ မိတ်ဆွေတို့အတွက် အသုံးဝင်ပြီး အထောက်အကူ ဖြစ်စေလိမ့်မယ်လို့ မျှော်လင့်ပါတယ်။ အားလုံး ကျန်းမာချမ်းသာကြပါစေ။

**Ye Myint Soe (Salai)**

Hornbill Technology

20.11.2022