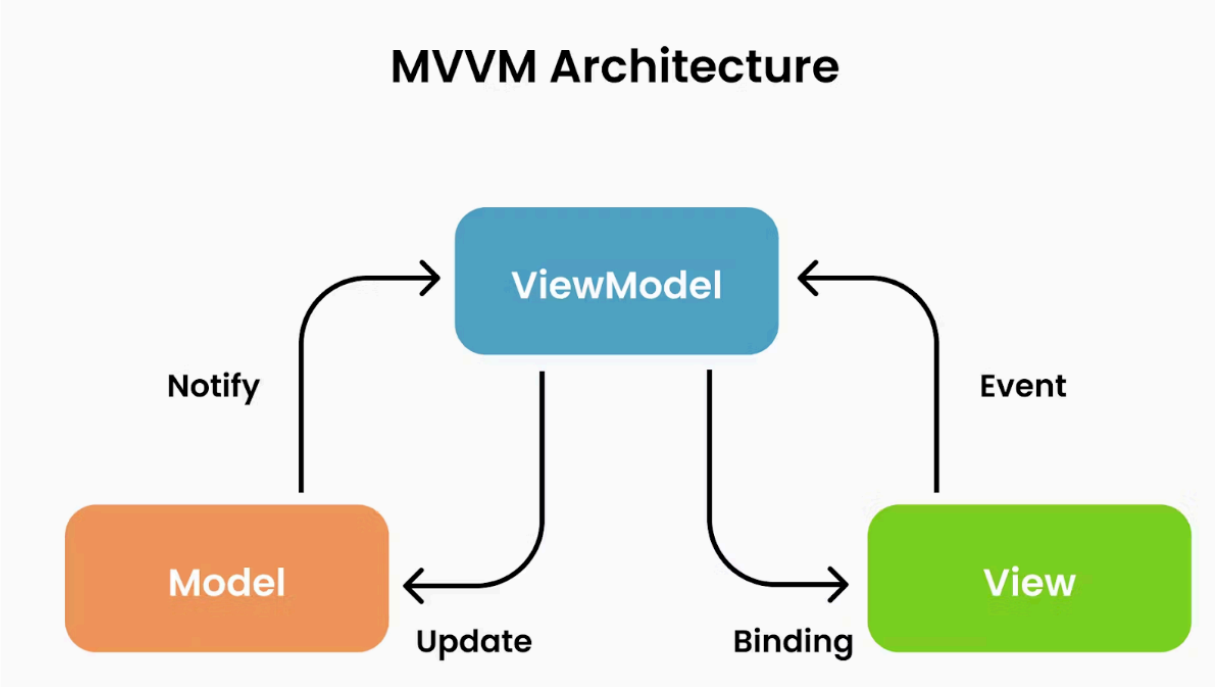
ch1_Vue_Architecture

What Is MVVM



MVVM（Model-View-ViewModel）架構是一種常見的軟體設計模式，通常用於前端開發，尤其是在桌面和移動應用程式中。它將應用程式分為三個主要部分：Model、View 和 ViewModel，每個部分負責不同的功能，從而促進代碼的模組化、可維護性和可擴展性

1. Model (資料)：Model 主要是應用程式的資料結構，並且負責資料的邏輯處理。在 Vue 中，Model 通常是 data 物件中所包含的資料，並且可能會涉及與 API 或資料庫的交互，但 Vue 本身並不處理這些交互，這通常是透過服務層或 Vuex 來實現。Class檔。

- **功能**：提供數據，並在數據變動時發出通知。它不關心如何顯示數據或如何與用戶交互，純粹是數據的來源。

- 範例：用戶資料、文章內容、商品清單等。

2. ViewModel (處理邏輯) ViewModel 是 MVVM 模式的核心，負責在 View 和 Model 之間進行數據的轉換與交互。它將 Model 中的數據轉換為 View 可以使用的格式，並將用戶的操作反映到 Model 中。ViewModel 通常充當 View 和 Model 之間的橋樑。

- 在 Vue 中，ViewModel 的角色主要由 data、computed 和 methods 完成，並且負責維護資料狀態、格式化資料和處理用戶的操作。
- 它會從 Model 中獲取數據並將其處理成 View 可以顯示的格式（例如，將時間戳格式化成易讀的日期）。
- 它會接收來自 View 的用戶操作（如點擊、輸入等），並將這些操作傳遞給 Model 進行處理。
- 它通常使用數據綁定技術，當數據變更時，View 會自動更新。

3. View (視圖)：View 就是使用者界面，是用戶可見的部分，展示 Model 資料並與 ViewModel 進行交互，不直接處理數據的變更。在 Vue 中，這通常是模板（HTML、DOM）部分，並通過資料綁定來呈現來自 ViewModel 的資料。

- 範例：用戶界面（UI），包括按鈕、輸入框、列表等。

How MVVM Work

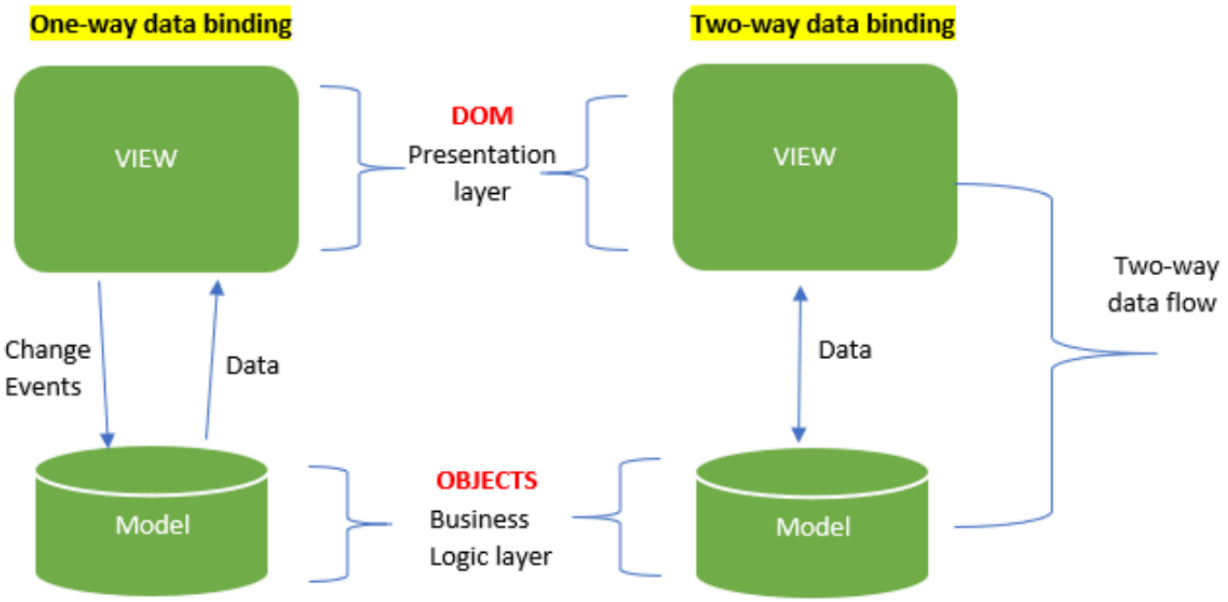
工作流程

1. **數據變化**：當 Model 中的數據發生變化（例如 API 返回新數據），它會通知 ViewModel。
2. **數據轉換**：ViewModel 會根據需要處理數據，將它轉換為適合顯示的格式，並更新自身的狀態。
3. **視圖更新**：ViewModel 通知 View 進行更新。由於 View 和 ViewModel 之間是雙向綁定的，View 會自動顯示更新後的數據。
4. **用戶操作**：用戶與 View 進行交互，觸發事件（例如點擊按鈕）。View 會將這些操作傳遞給 ViewModel。
5. **更新 Model**：ViewModel 接收到用戶操作後，將其傳遞給 Model，Model 根據需要執行業務邏輯並更新數據。

優勢

- **分離關注點**：MVVM 模式有助於將視圖邏輯和業務邏輯分離，使代碼更加清晰和易於維護。
- **數據綁定**：MVVM 使用數據綁定技術，能夠簡化 View 和 ViewModel 之間的交互。當數據變化時，View 會自動更新，無需手動操作 DOM。
- **可測試性**：由於 ViewModel 負責數據邏輯和交互邏輯，因此它可以單獨進行單元測試，而不需要依賴 UI。

資料綁定



單向資料綁定	雙向資料綁定
它透過單向渲染元素和元件來優化更新效能。	它透過渲染元素和元件雙向同步來優化更新效能
資料只是綁定到 DOM (bound to)，使用自訂程式碼觸發 DOM 變化	資料是從 DOM 綁定(bound from)的。當 DOM 改變時，數據也會改變。
它遵循資料模型中的靜態綁定	它遵循資料模型中的動態綁定
UI 可能需要一個事件處理程序來更新資料模型。	它遵循資料模型中的動態綁定

九大層

1. 模板層 (Template Layer) template

模板層是 Vue.js 應用中的 UI 層，主要負責定義 HTML 結構。模板通常與 Vue 元件中的 template 部分對應。Vue 的模板語法非常簡單直觀，支援插值語法、條件渲染、循環渲染等功能。

- **插值**：顯示動態內容，例如 {{ message }}。
- **指令**：控制 DOM 的行為，例如 v-if, v-for, v-bind 等。
- **事件綁定**：例如 v-on 來監聽用戶操作。

2. 數據層 (Data Layer) data

數據層是 Vue 應用中負責儲存和處理狀態的部分。在 Vue 中，data 用來儲存組件的狀態，而 Vue 的響應式系統會確保當數據改變時，視圖會自動更新。

- **響應式數據**：Vue 會自動追蹤依賴並在數據變更時更新 DOM。
- **computed 屬性**：用於根據數據進行衍生計算，並緩存結果。
- **watchers**：用來監視數據變化並執行額外的行為。
- 所以每次使用變數的時候，data會臨時建立物件並return回去。

3. 邏輯層 (Logic Layer) methods

邏輯層主要包括處理應用的業務邏輯，這些邏輯會寫在 Vue 元件的 methods 中。這些方法會響應用戶的操作、處理用戶輸入等。

- **Methods**：定義事件處理函數、數據更新的函數等。
- **Lifecycle Hooks**：如 mounted, created，鉤子函數用來處理元件的生命周期。

4. 樣式層 (Style Layer) style

Vue 也允許開發者在組件內部定義樣式。這些樣式通常寫在元件的 <style> 部分，可以進行作用域化，使得 CSS 只影響當前元件。

- **scoped**：用來限制樣式的作用範圍，避免影響到其他元件的樣式。

5.路由層 (Routing Layer) router-view router-link

Vue.js 有一個單獨的插件來管理應用的路由，即 Vue Router。它負責管理應用中的 URL 路徑與顯示的視圖組件之間的映射，從而實現單頁應用（SPA）中的頁面切換。

- **路由配置**：設置 URL 與對應的組件，處理瀏覽器的導航。
  - <router-view> 用來顯示當前路由匹配到的組件。
  - <router-link> 用來創建可導航的鏈接，實現路由的跳轉。
- **多嵌套路由**：允許組件內部有子路由，形成多層嵌套的 UI 結構。
  - 意味著在某些組件內部，你可以設置子路由。這樣可以實現多層嵌套的 UI 結構，讓應用的不同部分可以有獨立的路由系統。

6. 狀態管理層 (State Management Layer) Pinia

對於較大的應用，Vue.js 提供了 Vuex 作為狀態管理工具，來集中管理應用中的全局狀態。這使得在多個組件之間共享狀態變得更加簡單和清晰。

- **State**：應用的狀態。
- **Mutations**：同步修改狀態的方法。
- **Actions**：可以包含異步操作的行為。
- **Getters**：用來從狀態中衍生計算出的值。

7. 構建層 (Build Layer)

Vue.js 本身沒有強制的構建工具，但通常使用 Vue CLI 或 Vite 來構建應用程式。這些工具處理編譯、打包、壓縮等任務。

- **Vue CLI**：提供一個開箱即用的開發環境和構建工具。

Vite

：是基於現代瀏覽器支持的構建工具，Vue 3 推薦使用。

特性	Vue CLI	Vite
速度	啟動和編譯速度較慢，使用 webpack 打包	開發環境啟動速度快，使用原生 ESM 進行模塊加載
構建工具	基於 webpack，支持傳統打包過程	基於 Rollup，針對現代瀏覽器優化
配置靈活性	配置較為複雜，需要調整 webpack 配置	配置簡單，開箱即用
生產構建	生成的構建檔案經過優化，但較大	生成的構建檔案經過高效的優化，支持代碼分割
插件生態	插件生態豐富，支持 Vue、Vuex、Router 等	插件生態較為新穎，但不斷增長中
支持的項目	Vue 2 和 Vue 3	Vue 3，支持其他框架如 React 和 Preact
開發體驗	開發過程相對較慢，熱重載較慢	開發過程非常快速，熱重載即時反應

8. 插件層 (Plugin Layer)

Vue.js 的插件機制允許開發者擴展 Vue 的功能，像是 Vue Router、Vuex 或其他外部庫。通過插件，可以向 Vue 應用中注入新的功能或修改其行為。

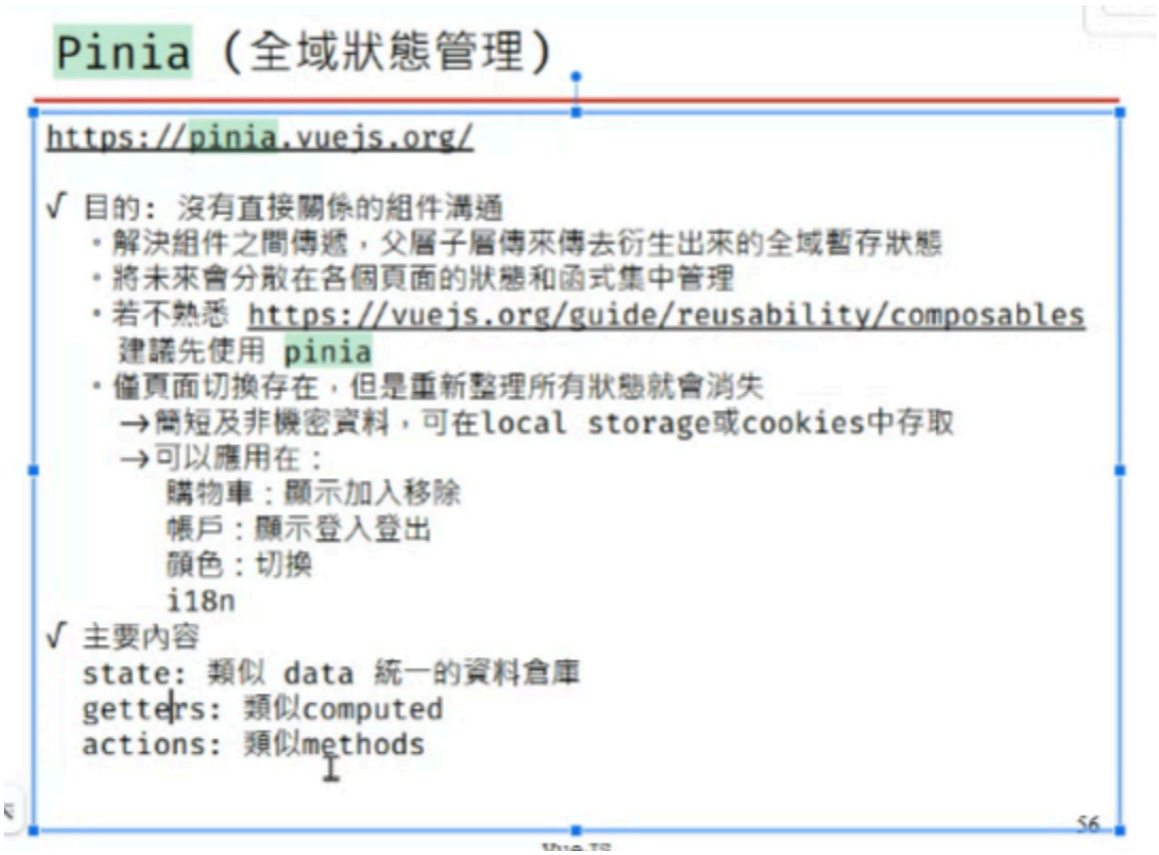
- **插件**：例如表單驗證、API 請求庫等。
- **混入 (Mixins)**：用來重複使用組件邏輯。

9. 服務層 (Service Layer) axios

服務層處理與後端 API 的交互，例如發送 HTTP 請求。這一層通常會將請求封裝成服務，以便其他部分能夠調用。

- **API 調用**：使用 axios 或 fetch 發送 HTTP 請求。
- **數據處理**：將從 API 返回的數據進行處理和格式化。

狀態管理層



1. State（狀態）

State 是應用的全局狀態，它包含了應用所需要的數據。State 是 Vuex 的核心，通常被用來儲存各種需要在多個組件之間共享的數據。

在組件中，我們可以通過 mapState 將 State 映射到組件的 computed 屬性中，從而使組件能夠讀取並顯示狀態數據。

2. Mutations（同步修改狀態的方法）

Mutations 是用來修改 state 的唯一方式。所有的狀態修改必須是同步的，因此它們通常是簡單的函數，並且每個 mutation 都應該只有一個目的：修改狀態。

```
// Vuex store 中的 mutations
const mutations = {
  increment(state) {
    state.count++
  },
  setUser(state, payload) {
    state.user = payload
  }
}
```

在組件中，我們可以通過 commit 方法來觸發 mutation。

`this.$store.commit('increment')` // 觸發 increment mutation

`this.$store.commit('setUser', { name: 'Jane Doe', age: 25 })` // 觸發 setUser mutation

3. Actions（可以包含異步操作的行為）

Actions 是用來執行異步操作的，並且可以在完成異步操作後調用 mutations 來修改狀態。Actions 是異步的，可以包含任何異步操作（例如：發送網絡請求）。

```
// Vuex store 中的 actions
const actions = {
  async fetchUser({ commit }) {
    const response = await fetch('/api/user')
    const userData = await response.json()
    commit('setUser', userData)  // 獲取數據後觸發 mutation 修改 state
  }
}
```

在組件中，我們可以通過 dispatch 方法來觸發 actions。

this.$store.dispatch('fetchUser') // 觸發 fetchUser action，並獲取用戶資料

4. Getters（衍生計算的值）

Getters 類似於 Vue 的計算屬性 (computed properties)，它們用來從 state 中衍生出某些計算後的值。Getters 是只讀的，並且通常用來過濾或計算 state 中的數據。

```
// Vuex store 中的 getters
const getters = {
  doubleCount(state) {
    return state.count * 2
  },
  userName(state) {
    return `${state.user.name} (Age: ${state.user.age})`
  }
}
```

在組件中，我們可以通過 mapGetters 將 getters 映射到組件的 computed 屬性中。

```
import { mapGetters } from 'vuex'

export default {
  computed: {
    ...mapGetters(['doubleCount', 'userName'])
  }
}
```

- **State**：存儲應用的數據，作為 Vuex 的核心。
- **Mutations**：同步修改 state 的方法。
- **Actions**：可以包含異步操作的行為，並觸發 mutations 更新狀態。
- **Getters**：用來從 state 中衍生計算出的值，類似於 Vue 的計算屬性。

Methods V.S. Computed

特性	`methods`	`computed`
觸發條件	每次調用時都會執行	只有當依賴的數據變化時才會重新計算
快取	不會快取，每次調用都會執行	會根據依賴數據進行快取
適用情況	處理事件、執行任意邏輯	用來處理基於 `data` 或其他屬性計算出來的屬性
性能	可能會影響性能，特別是有重複調用時	性能較好，因為會快取結果

1. methods（方法）

- **用途**：methods 用來定義函數或方法，這些方法可以在模板中或其他函數中被調用。這些方法可以執行任何類型的操作，並返回結果。
- **特點**：
  - 當你調用 methods 中的方法時，它會立即執行。
  - methods 中的函數會在每次觸發時重新計算，並且沒有快取。
  - 它適用於那些需要處理事件、執行計算、或其他邏輯操作的情況。

2. computed（計算屬性）

- **用途**：computed 用來定義基於數據（data）的計算屬性，這些屬性會自動根據其依賴的數據進行快取和更新。computed 更適合處理需要依賴數據進行計算的屬性，並且它只會在依賴的數據改變時重新計算。
- **特點**：
  - computed 是基於其依賴的數據來計算的，並且只有當依賴的數據變化時，計算屬性才會重新計算。
  - 它會被 Vue 自動快取，這意味著如果計算屬性所依賴的數據沒有改變，它不會重新計算。
  - 適用於需要從現有數據推導出新數據的情況。

3. 選擇使用場景

- **使用 methods**：
  - 需要在事件處理函數中執行邏輯（例如點擊按鈕時增加計數）。
  - 需要執行較複雜或異步的邏輯處理（例如調用 API 或進行數據處理）。
- **使用 computed**：
  - 當你需要基於已有數據計算出新數據時，並且希望自動快取結果（例如計算某個屬性的值，且只在相關數據變動時更新）。
  - 適合處理那些依賴於多個數據屬性並需要回傳新結果的情況（例如計算總價、篩選列表等）。

4. methods 計算新數據的缺點

使用 methods 來計算新數據有一些潛在的缺點，主要是性能和行為上的問題：

1. 每次渲染都會重新計算

- 當你在模板中使用 methods 時，每次重新渲染或更新時，這些方法會 **每次都被調用一次**，無論它們所依賴的數據是否有變化。
- 例如，假設 doubleCount() 會基於 count 計算並返回結果，如果 count 沒有改變，每次渲染時 doubleCount() 都會被調用，即使它返回的結果是相同的。這樣的行為對於計算比較複雜或者需要大量計算的數據會有性能上的影響，特別是當頁面頻繁重渲染時。

2. 不會快取計算結果

- methods 不會自動 **快取** 計算結果，即使輸入數據（data）沒有變化，每次調用方法都會重新執行計算。
- 與 computed 不同，computed 會根據依賴的數據進行 **快取**，只有在依賴的數據變化時才會重新計算。

3. 不適合用於基於數據的屬性

- 如果你的目的是為了計算屬性，並且這些屬性會頻繁更新（例如基於 data 中的某些屬性計算的結果），那麼使用 methods 來處理會比 computed 顯得不夠優雅。因為每次調用 methods 都會重新計算，而 computed 會自動根據依賴的數據更新。

4.何時使用 methods 計算新數據？

儘管有缺點，使用 methods 仍然適合某些情況：

- **需要執行動作而不僅是計算屬性**：例如，在事件處理中進行一些操作，而不僅僅是計算數據。
- **需要動態參數**：如果你的方法依賴於外部參數，這些參數在每次調用時都會改變，methods 是更合適的選擇。

5.使用 computed 的優勢

對比 methods，computed 在處理基於數據計算時有以下優勢：

1. **自動快取**：
    - 只有當依賴的數據（如 data）發生變化時，computed 屬性才會重新計算。這可以有效減少不必要的計算和提高性能。
2. **更符合語義**：

- 如果你的目的是為了計算一個屬性（例如 doubleCount），使用 computed 更符合其語義，因為它本質上是一個計算出來的屬性，而不是一個需要調用的函數。

6.總結

- **methods**：適合用於執行動作或事件處理，或者當你需要計算並且不關心計算的快取時使用。
- **computed**：適合用來計算屬性，並且當需要根據數據變化自動更新計算結果時，使用 computed 會更加高效且語義清晰。

如果只是基於 data 計算並返回一個新值，並且希望提高性能，使用 computed 是最佳選擇。

7.範例

- **computed** 是用來基於其他數據計算出新的數據。在這種情況下，當用戶選擇一個月份時，你需要基於選中的月份計算出該月份的天數，這是一個典型的 computed 屬性使用場景。computed 屬性會自動依賴於 data 中的選擇月份，並且只有當月份發生變化時，它才會重新計算天數。
- **methods** 是用來處理事件和執行動作的，通常用於改變數據或者處理事件。如果你用 methods 來實現，則每次調用時都會重新計算，並且沒有快取機制，這在這種情況下會顯得冗餘，尤其是當用戶沒有變更月份時。

```
<template>
  <div>
    <nav>
      <router-link to="/home">Home</router-link>
      <router-link to="/about">About</router-link>
    </nav>
    <router-view></router-view> <!-- 根據當前路由顯示不同內容 -->
  </div>
</template>
==========================================================
import { createRouter, createWebHistory } from 'vue-router'
import Home from './components/Home.vue'
import About from './components/About.vue'
```

```
const routes = [
  { path: '/home', component: Home },
  { path: '/about', component: About }
]

const router = createRouter({
  history: createWebHistory(),
  routes
})
export default router
```

```
// Vuex store 中的 state
const state = {
  count: 0,
  user: {
    name: 'John Doe',
    age: 30
  }
}

//在組件中，我們可以通過 mapState 將 State 映射到組件的 computed 屬性中，從而

import { mapState } from 'vuex'

export default {
  computed: {
    ...mapState(['count', 'user'])
  }
}
```

```
import { createStore } from 'vuex'

const store = createStore({
  state() {
```

```
    return {
      count: 0,
      user: {
        name: 'John Doe',
        age: 30
      }
    }
  },
  mutations: {
    increment(state) {
      state.count++
    },
    setUser(state, payload) {
      state.user = payload
    }
  },
  actions: {
    async fetchUser({ commit }) {
      const response = await fetch('/api/user')
      const userData = await response.json()
      commit('setUser', userData)
    }
  },
  getters: {
    doubleCount(state) {
      return state.count * 2
    },
    userName(state) {
      return `${state.user.name} (Age: ${state.user.age})`
    }
  }
})

export default store
```