

CIS 371 Web Application Programming

Pinia

App State Management



GRAND VALLEY
STATE UNIVERSITY®

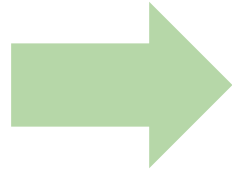
Lecturer: **Dr. Yong Zhuang**

What is Pinia?

Pinia is a state management solution created by Vue core team member Eduardo San Martin Morote, who also created Vue Router.



vuex



Pinia

What is state management?

Parent.vue

```
<template>
  <Child @childEvent="handleChildEvent" />
</template>

<script setup lang="ts">
import { ref } from 'vue';
import Child from './Child.vue';

const parentData = ref('');

const handleChildEvent = (data: string) => {
  parentData.value = data;
};
</script>
```

Emit data up via
events

Child

Parent

```
<template>
  <button @click="sendDataToParent">Send Data to Parent</button>
</template>

<script setup lang="ts">
import { defineEmits } from 'vue';

const emit = defineEmits(['childEvent']);

const sendDataToParent = () => {
  emit('childEvent', 'Data from child');
};
</script>
```

Child.vue

Demo

What is state management?

Parent.vue

```
<template>
  <Child :message="parentData" />
</template>

<script setup lang="ts">
import { ref } from 'vue';
import Child from './Child.vue';

const parentData = ref('Message from Parent');
</script>
```

Demo

```
<template>
  <div>{{ message }}</div>
</template>

<script setup lang="ts">
import { defineProps } from 'vue';

const props = defineProps({
  message: String
});
</script>
```

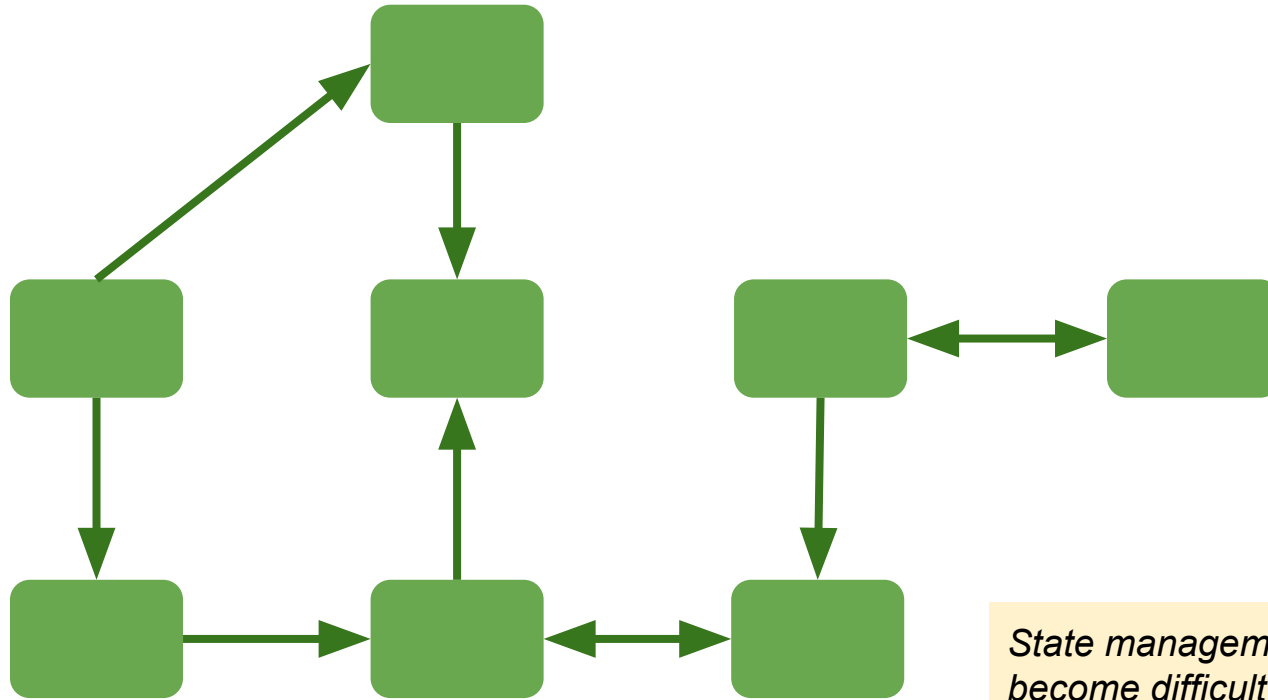
Child.vue

Pass data down as
props

Child

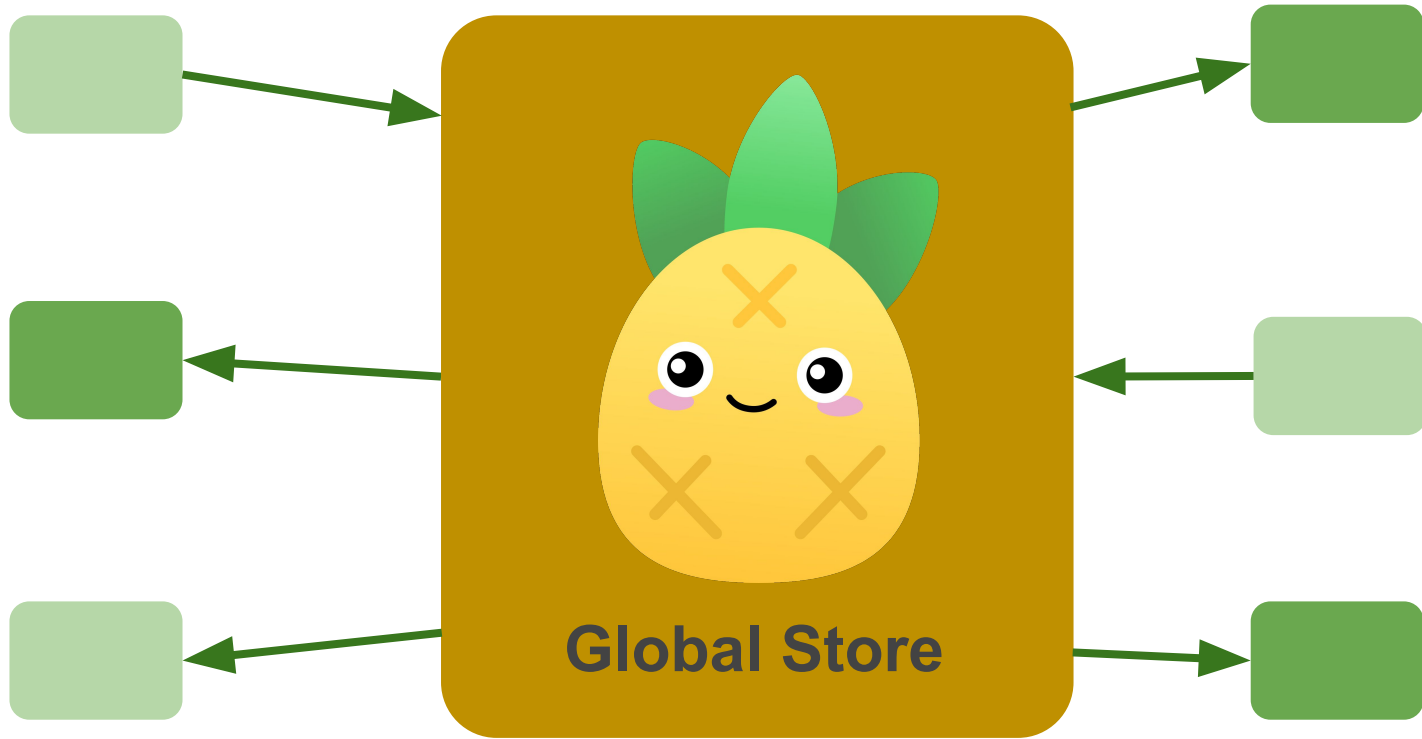
Parent

Large-scale SPA (single page application)



State management can quickly become difficult to maintain

Pinia: state management solution



Pinia is not a replacement for props and events



state management



props and events



Do I need to add Pinia to my project?

When should I use it?

Not necessary for

- Project with 5 - 10 components.
- Smaller projects with no shared state.
- Small demo projects.

Useful when:

- Project projected to grow
- Start using Pinia right away
- Almost no downside to using immediately



Why choose Pinia for state management?

Why choose Pinia for state management?

- Stores are as familiar as components. API designed to let you write well organized stores. (fewer new concepts to learn)
- Types are inferred, which means stores provide you with autocompletion even in JavaScript
- Pinia hooks into Vue devtools to give you an enhanced development experience in both Vue2 and Vue3.
- React to store changes to extend Pinia with transactions, local storage synchronization, etc.
- Build multiple stores and let your bundler code split them automatically.
- Pinia is extremely lightweight by design.

Installation

For a new project:

Official Vue3 project setup with interactive customization

```
o node → /workspaces $ npm init vue@3

Vue.js - The Progressive JavaScript Framework

✓ Project name: ... vue-pinia-app
✓ 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
```

In an existing project

```
npm install pinia
```

```
import { createApp } from "vue";
import App from "./App.vue";
import { createPinia } from "pinia";
```

```
createApp(App)
  .use(createPinia())
  .mount("#app");
```

App.vue

Pinia State

Define Pinia State

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

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return {};
  },
});
```

ItemStore.ts

Must be a function

Define Pinia State

```
export const useItemStore = defineStore("ItemStore", {  
  state: () => {  
    return {  
      category: "Electronics",  
      products: [  
        "Gamer's Delight Laptop",  
        "SmartTech Pro Phone",  
        "Rapid Dual USB-C Charger",  
      ],  
      inStock: true,  
    };  
  },  
});
```

ItemStore.ts

```
<script setup lang="ts">  
import { useItemStore } from "../stores/ItemStore";  
  
const itemStore = useItemStore();  
console.log(itemStore.category);  
</script>
```

AnyVueComponent.vue

Access Pinia State

```
<script setup lang="ts">
import { useItemStore } from "../stores/ItemStore";

const itemStore = useItemStore();
console.log(itemStore.category);
</script>
```

AnyVueComponent.vue

category

\$onAction

\$patch

(property) category: string

x

State is TypeSafe

Access Pinia State

AnyVueComponent.vue

```
<script setup lang="ts">
import { useItemStore } from "../stores/useItemStore";

const { category } = useItemStore();
console.log(category);
</script>
```

Can de-structure state from store

*category is **no longer reactive***

Access Pinia State

AnyVueComponent.vue

```
<script setup>
import { useItemStore } from "../stores/useItemStore";
import { storeToRefs } from "pinia";

const { category } = storeToRefs(useItemStore());
console.log(category.value);
</script>
```

convert store to **refs** to maintain reactivity

Can de-structure state from store

Update Pinia State

AnyVueComponent.vue

```
<script setup>
import { useItemStore } from "../stores/useItemStore";
import { storeToRefs } from "pinia";

const { category } = storeToRefs(useItemStore());
category.value = "Groceries";
</script>
```

State can be directly updated

AnyVueComponent.vue

```
<script setup lang="ts">
import { useItemStore } from "../stores/useItemStore";

const itemStore = useItemStore();
itemStore.category = "Groceries";
</script>
```

If you don't de-structure, you don't need .value

Two-way Binding

```
<script setup lang="ts">
import { useItemStore } from "../stores/ItemStore";
import { storeToRefs } from "pinia";
const { category } = storeToRefs(useItemStore());
</script>

<template>
  <input v-model="category" type="text" />
</template>
```

AnyVueComponent.vue

Demo

v-for

ItemStore.ts

```
import { defineStore } from "pinia";
import products from "../data/products.json";

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  }
});
```

```
<script setup lang="ts">
import { useItemStore } from "../stores/ItemStore";
const items = useItemStore();
</script>

<template>
  <ul v-for="(item, idx) in items.products" :key="idx" :item="item">
    <li>{{ item.name }}</li>
  </ul>
</template>
```

AnyVueComponent.vue

Demo

Pinia Actions

Pinia Actions

```
import { defineStore } from "pinia";
import products from "../data/products.json";

export const useItemStore = defineStore("ItemStore", {
  state: () => {
    return { products };
  },

  actions: {
    fill() {},
  },
});
```

ItemStore.ts

Define actions as methods on the actions option

Pinia Actions

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

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: [] };
  },

  actions: {
    async fill() {
      const res = await fetch(
        "https://randomuser.me/api?results=5&nat=gb,fr"
      );
      this.users = await res.json();
    },
  },
});
```

UserStore.ts

```
actions: {
  async fill() {
    const res = await fetch(
      "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"
    );
    this.users = await res.json();
    this.someOtherAction();
  },
  async someOtherAction() {},
},
});
```

Access state with `this`

Access other actions with `this`

Pinia Actions

user.ts

```
type User = {  
  name: {  
    first: string;  
    last: string;  
    title: string;  
  };  
  email: string;  
  picture: {  
    large: string;  
    medium: string;  
    thumbnail: string;  
  };  
};  
type RandomUser = {  
  results: Array<User>;  
};  
export type { User, RandomUser };
```

UserStore.ts

```
import { defineStore } from "pinia";  
import { RandomUser } from "../types/user";  
  
export const useUserStore = defineStore("UserStore", {  
  state: () => {  
    return { users: { results: [] } as RandomUser };  
  },  
  
  actions: {  
    async fill() {  
      const res = await fetch(  
        "https://randomuser.me/api?results=5&nat=gb,fr&inc=name,email,picture"  
      );  
      this.users = await res.json();  
      console.log(this.users);  
    },  
  },  
});
```

Pinia Actions

AnyVueComponent.vue

```
<script setup lang="ts">
import { useUserStore } from "../stores/UserStore";
const userStore = useUserStore();
userStore.fill();
</script>

<template>
  <ul v-for="(u, idx) in userStore.users.results" :key="idx" :u="u">
    <li>{{ u.name.first }}. {{ u.name.last }}</li>
  </ul>
</template>
```

Demo

Pinia Actions

```
<script setup lang="ts">
import { useUserStore } from "../stores/UserStore";
const { fill } = useUserStore();
fill();
</script>
```

de-structure

```
<script setup lang="ts">
import { useUserStore } from "../stores/UserStore";
import { storeToRefs } from "pinia";
const { fill } = storeToRefs(useUserStore());
fill();
</script>
```

won't work

Demo

Pinia Actions: Example

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

export const useCartStore = defineStore("CartStore", {
  state: () => {
    return {
      items: [],
    };
  },
  actions: {
    addItem(itemId, count) {
      // set the count for the proper item in the state above
    },
  },
});
```

CartStore.ts

```
<button @click="addItem(product.id, $event)">Add Product to Cart</button>
```

Item.vue

Useful for updating state based on user interaction

Summary

What is state management?

What is Pinia?

- State,
- Actions,

