# CIS 371 Web Application Programming

## Pinia

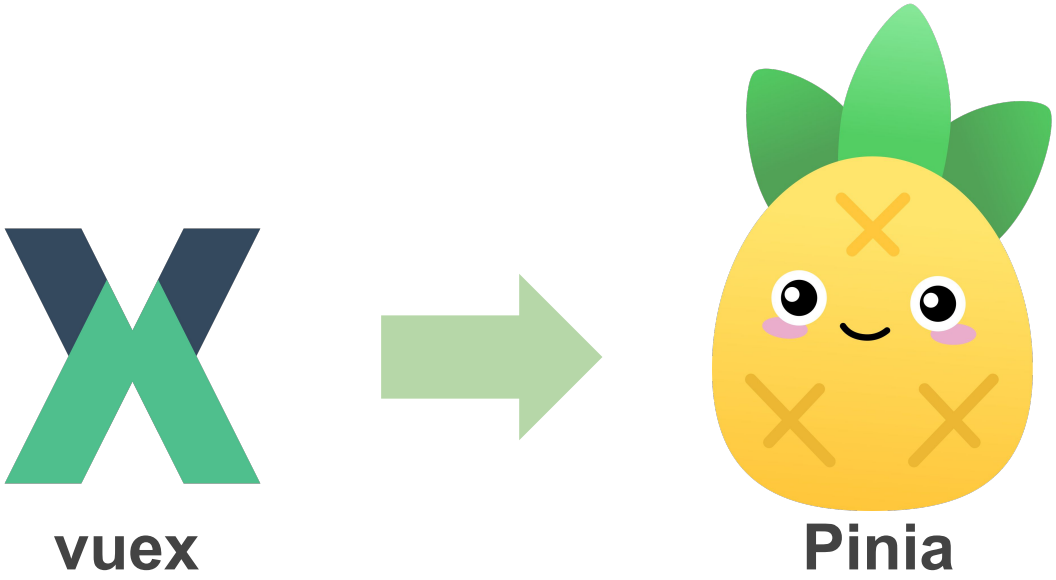
### App State Management


**Lecturer: Dr. Yong Zhuang**

Based on the original version by Daniel Kelly.

# What is Pinia?

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



# What is state management?

*Parent.vue*

```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>
```


```vue
<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*

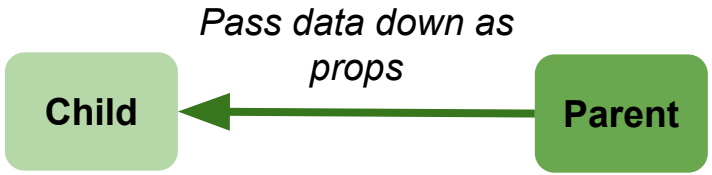
# What is state management?

*Parent.vue*

```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>
```



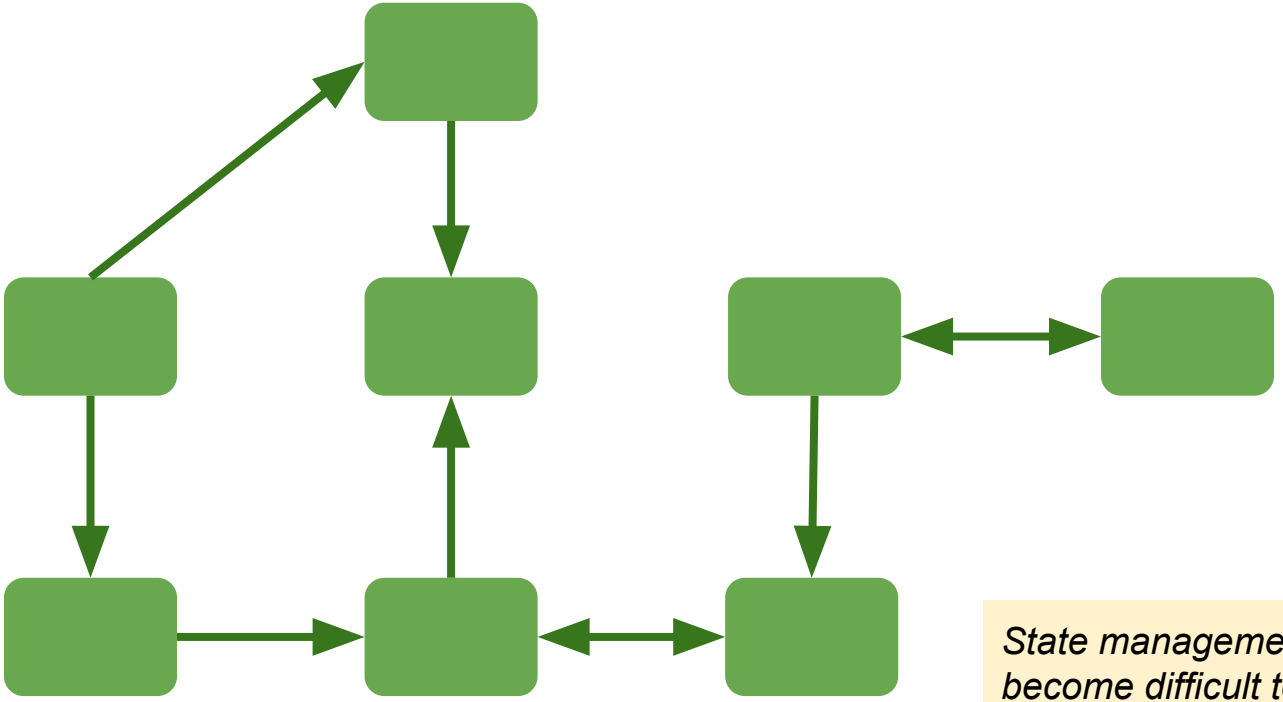

```vue
<template>
  <div>{{ message }}</div>
</template>

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

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

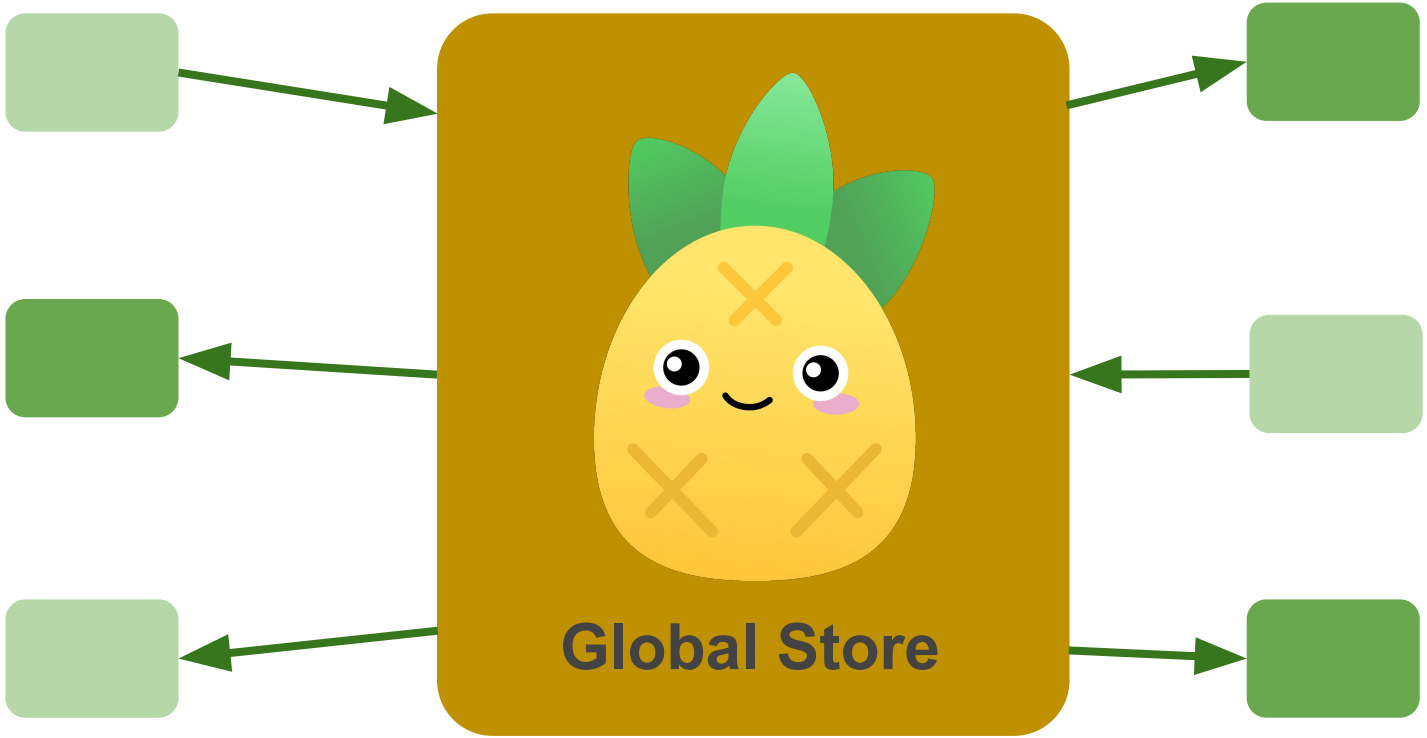
*Child.vue*

# 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

```
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*

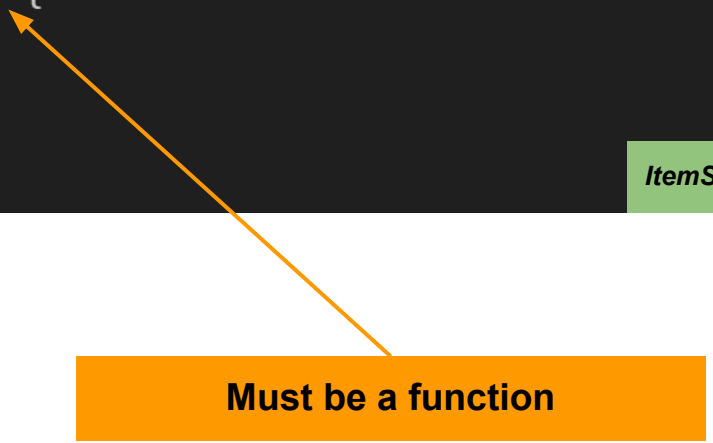
# Pinia State

# Define Pinia State

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

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

*ItemStore.ts*



# Define Pinia State

```ts
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*

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

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

*AnyVueComponent.vue*

# Access Pinia State

*AnyVueComponent.vue*

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

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

```
category
$onAction
$patch
```

```
(property) category: string
```

**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*

```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*

```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*

```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

```vue
<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*

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

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

*AnyVueComponent.vue*

```vue
<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>
```

Demo

# Pinia Actions

# Pinia Actions

*ItemStore.ts*

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

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

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

**Define actions as methods on the actions option**

# Pinia Actions

*UserStore.ts*

```ts
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&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*

```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*

```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

*CartStore.ts*

```
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
    },
  },
});
```

*Item.vue*

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

*Useful for updating state based on user interaction*

# Pinia Getters

# What are Pinia Getters?

Equivalent of computed props on a component

**Must explicitly type the return**

**Access state with `this`**

*UserStore.ts*

```ts
import { defineStore } from "pinia";
import { RandomUser } from "../types/user";

export const useUserStore = defineStore("UserStore", {
  state: () => {
    return { users: {} as RandomUser };
  },
  getters: {
    count(): number {
      return this.users.results.length;
    },
  },
  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();
    },
  },
});
```

# Pinia Getters

**Access state with `state`**

```
getters: {
  count(state): number {
    return state.users.results.length;
  },
},
```

**No need to explicitly type return**

```
getters: {
  count: (state) => state.users.results.length,
},
```

*single line arrow functions*

**Access other getters on `this`**

```
getters: {
  count: (state) => state.users.results.length,
  doubleCount(): number {
    return this.count * 2;
  },

  findUserByFirstName: (state) => (first: string) => {
    return state.users.results.find(
      (user: User) => user.name.first === first
    );
  },
},
```

**Accept arguments by returning a function**

# Access Getters

*as a property*

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

*de-structure*

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

*Can de-structure getters from store but must use `storeToRefs`*

Demo