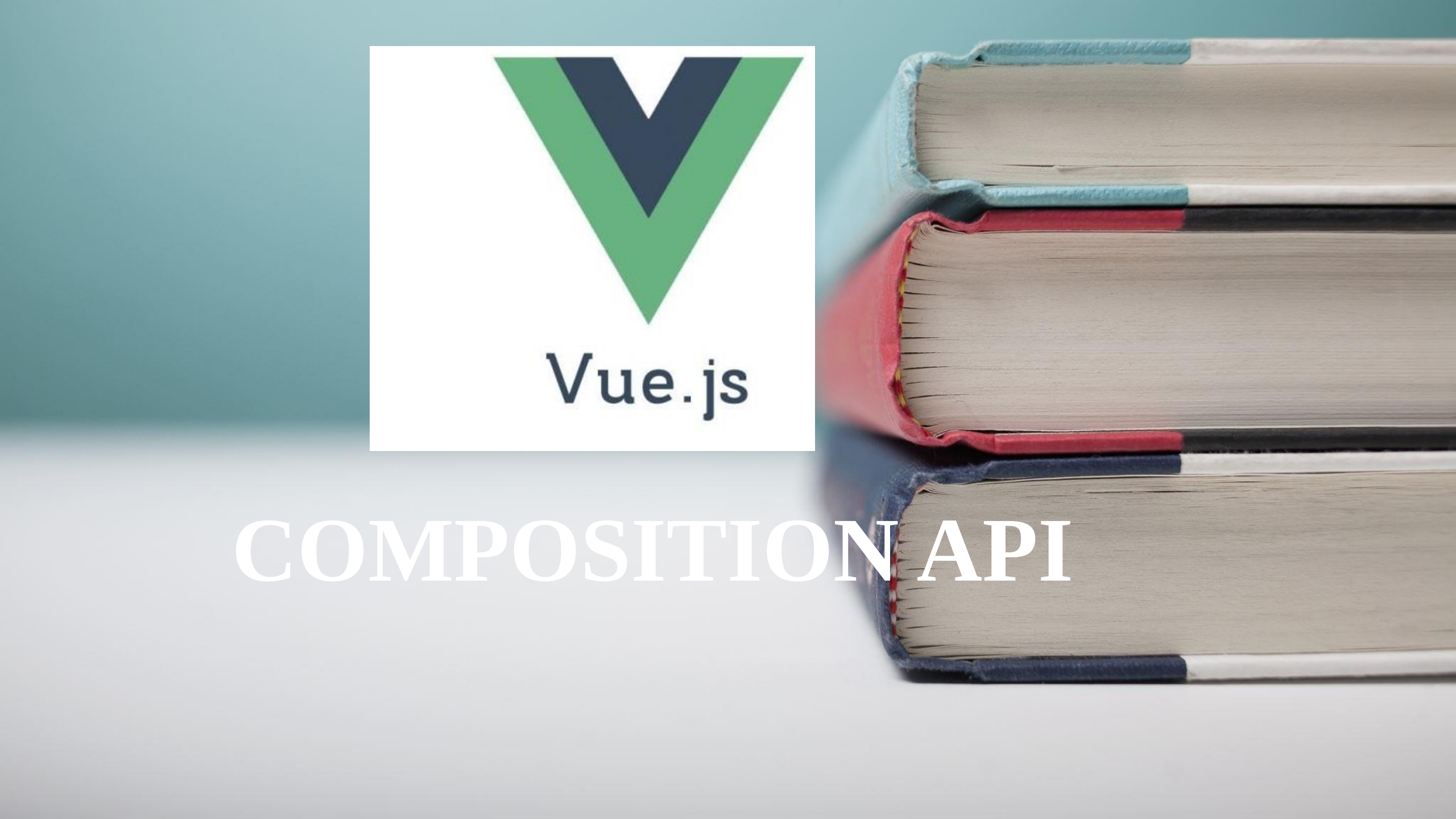

COMPOSITION API

NỘI DUNG CHÍNH

I Giới thiệu

II Reactive()

III Ref()

IV <Script setup>

1.Giới thiệu về tính phản ứng trong Vue 3

- **Tính Phản Ứng là gì?**
- Tính phản ứng là một tính năng cốt lõi trong Vue 3 cho phép ứng dụng tự động phản hồi khi dữ liệu thay đổi.
- Giúp tạo ra các giao diện động, dựa trên dữ liệu, nơi mà mọi thay đổi trong dữ liệu sẽ được phản ánh ngay lập tức trên UI.

1.Giới thiệu về tính phản ứng trong Vue 3

- **Các thành phần chính**
- **Dữ liệu Phản Ứng (Reactive Data):** Các đối tượng theo dõi phụ thuộc và kích hoạt cập nhật khi có thay đổi.
- **Ref:** Một wrapper phản ứng cho một giá trị đơn lẻ, có thuộc tính **.value.**
- **Thuộc Tính Tính Toán (Computed Properties):** Trạng thái dẫn xuất được cập nhật dựa trên các phụ thuộc phản ứng.
- **Watchers:** Cho phép thực hiện các hành động phụ khi dữ liệu phản ứng thay đổi.

2. Tạo dữ liệu phản ứng với `reactive()`

- **Cú pháp**

```
import { reactive } from 'vue';

const state = reactive({
  count: 0,
  message: 'Xin chào Vue!'
});
```

- **Hàm reactive()** chuyển đổi một đối tượng thành một proxy phản ứng, làm cho tất cả các thuộc tính của nó trở nên phản ứng.
- Mọi thay đổi đối với **state.count** hoặc **state.message** sẽ tự động cập nhật mọi liên kết UI.

3. Sử dụng `ref()`

- **Tại sao sử dụng `ref()`**
  - **`ref()`** được sử dụng để tạo một tham chiếu phản ứng cho các giá trị nguyên thủy như số, chuỗi, boolean và đối tượng
  - Cung cấp một cách đơn giản để theo dõi và cập nhật giá trị mà UI có thể phản hồi lại.
- **Cú pháp:**

```
import { ref } from 'vue'

const count = ref(0)
```

3. Sử dụng `ref()`

- **`ref()`** nhận vào một giá trị và trả về một đối tượng với thuộc tính .value, chứa giá trị gốc.
- **Ví dụ:** count.value sẽ trả về 0 (giá trị ban đầu).

```
const count = ref(0)

console.log(count) // { value: 0 }
console.log(count.value) // 0

count.value++
console.log(count.value) // 1
```

3. Sử dụng `ref()`

- Truy cập giá trị

```
console.log(count.value);  // Output: 0
```

- Cập nhật giá trị

```
count.value = 5;
console.log(count.value);  // Output: 5
```

- Khi giá trị của **ref** thay đổi, Vue sẽ tự động cập nhật UI để phản ánh thay đổi này.

3. Sử dụng `ref()`

- **Ví dụ:**

```
<template>
  <div>
    <p>Số lần nhấn: {{ count.value }}</p>
    <button @click="increment">Nhấn để tăng</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';

const count = ref(0);

function increment() {
  count.value++;
}
</script>
```

3. Sử dụng `ref()’

- Để truy cập đến thành phần template, khai báo và trả về từ hàm **setup()**

```
import { ref } from 'vue'

export default {
  // `setup` is a special hook dedicated for the Composition API.
  setup() {
    const count = ref(0)

    // expose the ref to the template
    return {
      count
    }
  }
}
```

```
<div>{{ count }}</div>
```

3. Sử dụng `ref()’

- Sử dụng **`ref()`** với đối tượng
- Cú pháp

```
const person = ref({
  name: 'John',
  age: 30
});
```

- Truy cập các thuộc tính thông qua **`.value`**

```
person.value.name = 'Doe';
console.log(person.value.name);  // Output: Doe
```

Lưu ý: Dù sử dụng đối tượng, vẫn cần truy cập thông qua `.value`

3. Sử dụng `ref()`

- **Lưu ý:** Không cần phải thêm **.value** khi sử dụng **ref** trong template. Để thuận tiện, **ref** sẽ tự động được mở gói khi sử dụng bên trong template (với một vài lưu ý).
- Có thể thay đổi **ref** trực tiếp trong trình xử lý sự kiện:

```
<button @click="count++">
  {{ count }}
</button>
```

- Đối với logic phức tạp hơn, có thể khai báo các hàm biến đổi tham chiếu trong cùng phạm vi và hiển thị chúng dưới dạng các phương thức cùng với trạng thái:

3. Sử dụng `ref()`

```
import { ref } from 'vue'

export default {
  setup() {
    const count = ref(0)

    function increment() {
      // .value is needed in JavaScript
      count.value++
    }

    // don't forget to expose the function as well.
    return {
      count,
      increment
    }
  }
}
```

- *Các phương thức được hiển thị sau đó có thể được sử dụng làm trình xử lý sự kiện:*

```
<button @click="increment">
  {{ count }}
</button>
```

4. `ref()` với reactive()

- **Sự khác biệt chính:**
  - ref(): Thích hợp cho giá trị nguyên thủy và có thể sử dụng cho đối tượng.
  - reactive(): Biến một đối tượng thành proxy phản ứng, thích hợp cho các đối tượng phức tạp.
- **Khi nào nên sử dụng ref()?**
  - Khi bạn làm việc với giá trị đơn lẻ hoặc muốn đảm bảo rằng các thay đổi nhỏ trong đối tượng không bị mất đi tính phản ứng.

4. `ref()` với reactive()

- **Ví dụ:**

```
import { ref, reactive } from 'vue';

const count = ref(0);
const person = reactive({
  name: 'John',
  age: 30
});

function updatePerson() {
  person.name = 'Doe';
  count.value++;
}
```

4. <script setup>

- <script setup> là cú pháp đơn giản hóa trong Vue 3, giúp dễ dàng viết mã cho các thành phần (components) bằng Composition API.
- Tất cả các biến và hàm khai báo bên trong <script setup> sẽ tự động có sẵn trong template mà không cần phải trả về như trước.

4. <script setup>

- **Ưu điểm của <script setup>**

✓ Tối ưu hóa hiệu suất:

- Được biên dịch nhanh hơn và hiệu quả hơn.
- Mã ngắn gọn hơn:
- Không cần sử dụng return hoặc các đối tượng setup để khai báo các biến và hàm.

✓ Dễ đọc và bảo trì:

- Giảm bớt sự phức tạp trong mã nguồn và làm cho mã dễ đọc hơn.

4. <script setup>

- Cú pháp:

```vue
<script setup>
import { ref } from 'vue'

const count = ref(0)

function increment() {
  count.value++
}
</script>

<template>
  <button @click="increment">
    {{ count }}
  </button>
</template>
```

4. <script setup>

- So sánh **<script setup> và setup():**
- **Setup()**

```
<script>
import { ref } from 'vue';

export default {
  setup() {
    const count = ref(0);
    const increment = () => {
      count.value++;
    };
    return { count, increment };
  }
}
</script>
```

<script setup>

```
<script setup>
import { ref } from 'vue';

const count = ref(0);
const increment = () => {
  count.value++;
};
</script>
```

4. <script setup>

- Kết hợp các tính năng khác của Vue
- **Props**

```
<script setup>
defineProps(['title']);
</script>

<template>
  <h1>{{ title }}</h1>
</template>
```

Emit

```
<script setup>
const emit = defineEmits(['update']);

const updateValue = () => {
  emit('update', newValue);
};
</script>
```

4. <script setup>

- **Ví dụ:** Form nhập liệu

```
<template>
  <form @submit.prevent="handleSubmit">
    <input v-model="name" placeholder="Nhập tên của bạn" />
    <button type="submit">Gửi</button>
  </form>
</template>

<script setup>
import { ref } from 'vue';

const name = ref('');
const handleSubmit = () => {
  console.log('Tên đã nhập:', name.value);
};
</script>
```

4. <script setup>

- **Lưu ý:**
- Không thể sử dụng **this**: **<script setup>** không hỗ trợ **this**, vì toàn bộ cú pháp là **functional**.
- Không hỗ trợ **Hoisting**: Tất cả các biến và hàm phải được khai báo trước khi sử dụng.
- Hạn chế việc chia sẻ mã: Nếu có mã cần sử dụng lại trong nhiều component, <script setup> có thể không phải là lựa chọn tốt nhất so với setup().


the
end