

# XÂY DỰNG GIAO DIỆN TƯƠNG TÁC BACKEND

## BÀI 5: DATA BINDING TRONG VUEJS

- ◎ Kết thúc bài học này bạn có khả năng
  - Tìm hiểu Data binding
  - Nắm được khái niệm Reactivity
  - Hiểu được Class và Style binding



## Phần I: Data binding và Reactivity trong VueJS

- ❖ Data binding

- ❖ Reactivity

## Phần II: Class và Style binding

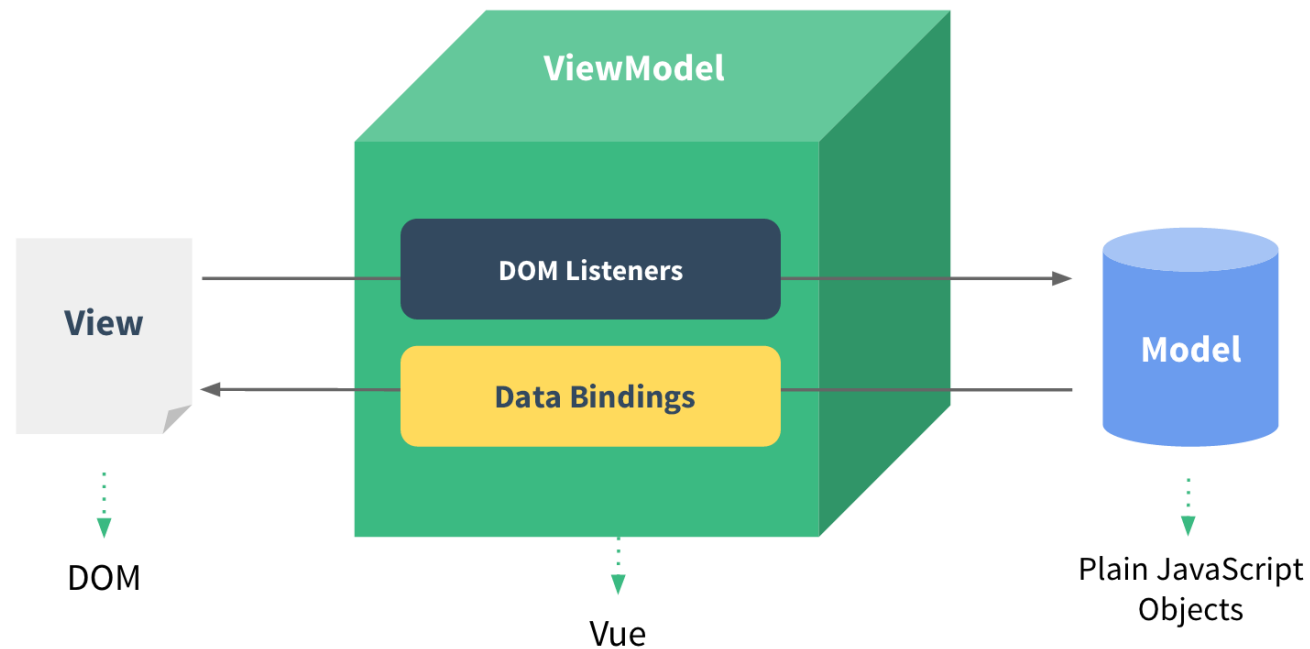
- ❖ Classes Binding (Liên kết các lớp)

- ❖ Binding inline styles (Ràng buộc kiểu nội tuyến)



**BÀI 5:**  
**DATA BINDING TRONG VUEJS**

**PHẦN I: DATA BINDING – REACTIVITY**  
**TRONG VUEJS**



- ❑ Data binding là một khái niệm cốt lõi, cho phép liên kết dữ liệu trong ứng dụng với giao diện người dùng một cách tự động. Điều này có nghĩa là khi dữ liệu thay đổi, giao diện sẽ tự động cập nhật để phản ánh sự thay đổi đó và ngược lại.

Có hai cách bind data trong VueJs: **One-way data binding** (một chiều) và **Two-way data binding** (hai chiều).

❑ **One-way data binding:**

*Dữ liệu thay đổi => DOM render lại dữ liệu*

- ❖ Liên kết data trực tiếp từ code javascript với DOM.
- ❖ Sử dụng ***v-bind*** để gắn dữ liệu động vào.

```
<template>
  <p>
    Name: <input type="text" :value="name">
    <!-- Name: <input type="text" v-bind:value="name"> -->
  </p>
  <a class = 'btn btn-primary' :href="url">click me</a>
</template>
<script>
export default {
  data() {
    return {
      name: 'Vue',
      url: 'https://www.google.com/',
    }
  }
}
</script>
```

Name:

[click me](#)



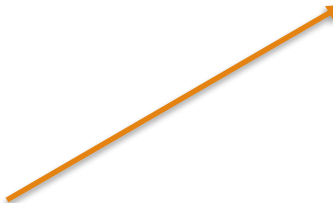
- ❑ **Props** trong One-way data binding: **props** là một luồng dữ liệu một chiều truyền từ component cha xuống component con. Component con sẽ nhận và sử dụng dữ liệu này thông qua việc khai báo props.
- ❑ Vue.js cho phép xác thực **props** bằng cách kiểm tra kiểu dữ liệu, yêu cầu bắt buộc, và giá trị mặc định. Điều này giúp đảm bảo component luôn nhận được dữ liệu theo đúng yêu cầu.

```
props: {  
  title: {  
    type: String,  
    required: true  
  },  
  content: {  
    type: String,  
    default: "Nội dung mặc định"  
  }  
}
```

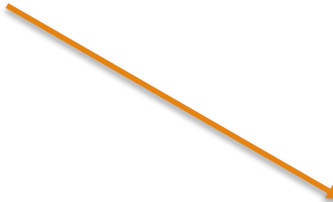


# ONE-WAY DATA BINDING VỚI PROPS

- ❑ **Props** trong One-way data binding:  
Truyền đối tượng hoặc mảng thông qua props
- ❑ Có thể truyền các kiểu dữ liệu phức tạp như **mảng** hoặc **đối tượng** qua props. Cú pháp:



```
const props = defineProps({  
  propertyName: Array  
});
```



```
const props = defineProps({  
  propertyName: Object  
});
```

**Two-way data binding:** đồng bộ tương tác qua lại giữa view và modal. Như vậy, trạng thái sẽ được cập nhật bất cứ khi nào template thay đổi và ngược lại, giúp tăng tốc đáng kể quá trình phát triển.

*Dữ liệu thay đổi => DOM render lại dữ liệu và ngược lại*

☐ Sử dụng directive: ***v-model***

**v-model:** Vue đã dựng sẵn các hàm API tương ứng. Khi thực hiện chỉ cần truyền data vào v-model của input đó.

```
<template>
  <input type="text" v-model="userName">
  <p>Xin chào: {{ userName }}</p>
</template>
<script>
export default {
  data() {
    return {
      userName: 'VueJS'
    }
  }
}
</script>
```

Nội dung trong input thay đổi dẫn đến nội dung trong phần tử p cũng thay đổi theo

VueJS

Xin chào: VueJS

VueJS Framework

Xin chào: VueJS Framework

*V-model sẽ được học kỹ hơn ở những bài học sau*

- ❑ Trước khi đi vào phần **Reactivity**, chúng ta cần nắm được khái niệm **Composition API**
- ❑ **Composition API** là một cách tiếp cận mới trong Vue.js 3 để xây dựng các component, cung cấp một giải pháp linh hoạt hơn so với Options API (cách truyền thống). Nó cho phép tổ chức và tái sử dụng logic một cách rõ ràng và có tổ chức hơn, đặc biệt là khi làm việc với các component phức tạp.
- ❑ Các API chính trong Composition API:
  - ❖ **setup**: là một hàm đặc biệt được gọi trước khi component được khởi tạo. Dữ liệu và logic khai báo trong setup sẽ được trả về và có thể được sử dụng trong template.
  - ❖ **ref** và **reactive**: tạo các đối tượng phản ứng (được giới thiệu chi tiết trong bài học này)
  - ❖ **computed**: tạo ra các giá trị phụ thuộc dựa trên các giá trị reactive khác.
  - ❖ **watch**: theo dõi sự thay đổi của các reactive state và thực thi một hàm khi thay đổi được

## ❑ **Reactivity**: Tính phản ứng

Vue.js sử dụng Proxy API để tạo ra các đối tượng phản ứng, cho phép theo dõi sự thay đổi trong dữ liệu và tự động cập nhật giao diện người dùng.

### ❑ Có 2 Reactivity API:

- ❖ **ref**: Được sử dụng để tạo một đối tượng reactive đơn giản cho các **giá trị** cơ bản.
- ❖ **reactive**: Tạo một đối tượng reactive từ một **đối tượng** JavaScript thông thường

## ❑ Bảng so sánh **Ref** và **Reactive**

Tiêu chí	Ref	Reactive
Mục đích sử dụng	Sử dụng để tạo reactive value đơn giản hoặc primitive value.	Sử dụng để tạo reactive object hoặc array.
Kiểu dữ liệu	Bất kỳ kiểu dữ liệu nào (primitive hoặc object).	Object hoặc Array.
Kiểu trả về	Trả về một object với thuộc tính <b>.value</b> lưu trữ giá trị thực.	Trả về một reactive proxy của object/array.
Cách truy cập giá trị	Truy cập giá trị thông qua <b>.value</b> .	Truy cập trực tiếp các thuộc tính của object/array
Hạn chế	Cần truy cập giá trị thông qua <b>.value</b> , có thể hơi phức tạp nếu bạn quen với việc sử dụng trực tiếp.	Khó khăn trong việc theo dõi và debug các thay đổi nếu object quá lớn.

❑ **Ref:** là một trong những API cơ bản trong Vue để tạo phản ứng tĩnh. Vue sẽ tạo ra một object có thuộc tính **.value** để lưu trữ giá trị thực. Bất cứ khi nào giá trị **.value** thay đổi, Vue sẽ tự động cập nhật DOM nếu giá trị đó được sử dụng trong template. Trong **Composition API**, cách được khuyến nghị để khai báo trạng thái phản ứng là sử dụng **Ref**.

❑ **Cách sử dụng Ref:**

1. Import ref từ Vue:

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

2. Tạo một biến reactive với ref:

```
const count = ref(0);
```



**count** là một object, và giá trị thực của nó được lưu trữ trong thuộc tính **.value**

## ❑ Cách sử dụng Ref (tiếp theo):

### 3. Truy cập và cập nhật giá trị:

```
console.log(count.value); // 0  
count.value = 5;  
console.log(count.value); // 5
```

### 4. Sử dụng trong template:

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

+



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



## ❑ Dùng Ref với các kiểu dữ liệu:

❖ **Primitive values:** (Số, chuỗi, boolean)

```
const number = ref(5)
const name = ref('John');
const isActive = ref(false);
const flag = ref(null);
```

❖ **Objects/Arrays:** Nếu object hoặc array không cần phản ứng ở từng thuộc tính riêng lẻ, **ref** vẫn là lựa chọn tốt.

```
const user = ref({
  name: 'Alice',
  age: 30
});

user.value.age = 31; // Cập nhật giá trị thông qua `.value`
```

❑ **Reactive():** là một API quan trọng cho phép tạo reactive state cho các object hoặc array.

Khi một object được bọc bởi reactive, tất cả các thuộc tính của object đó sẽ trở thành reactive, nghĩa là Vue sẽ theo dõi các thay đổi và tự động cập nhật giao diện người dùng khi các thuộc tính đó thay đổi.

## ❑ Cách sử dụng Ref:

1. Import reactive từ Vue:

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

2. Tạo một reactive object:

state là một object reactive

```
const state = reactive({  
  count: 0,  
  user: {  
    name: 'John',  
    age: 25  
  }  
});
```

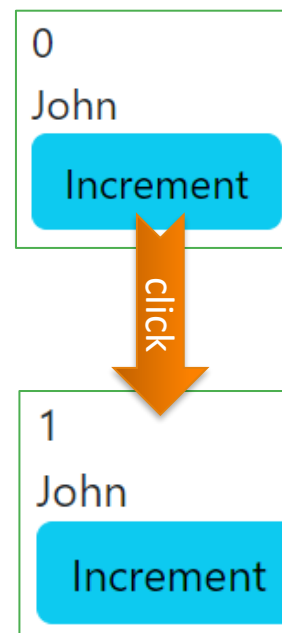
## ❑ Cách sử dụng Reactive (tiếp theo):

### 3. Truy cập và cập nhật giá trị:

```
console.log(state.count); // 0
state.count++;
console.log(state.count); // 1
```

### 4. Sử dụng trong template:

```
<template>
  <div>{{ state.count }}</div>
  <div>{{ state.user.name }}</div>
  <button @click="increment">Increment</button>
</template>
```



+

```
<script>
import { reactive } from 'vue';
export default {
  setup() {
    const state = reactive({
      count: 0,
      user: {
        name: 'John',
        age: 25
      }
    });

    const increment = () => {
      state.count++;
    };

    return { state, increment };
  }
};
</script>
```

## ❑ Reactive: Các tính năng nâng cao

- ❖ **Nested Reactivity** (Tính chất phản ứng lồng nhau): Vue sẽ làm cho toàn bộ object và các object con của nó trở thành reactive. Ví dụ:

```
const state = reactive({
  user: {
    name: 'Sunny',
    address: {
      city: 'New York'
    }
  }
});

state.user.address.city = 'Los Angeles'; // Vue sẽ theo dõi sự thay đổi này.
```

## ❑ Reactive: Các tính năng nâng cao

- ❖ **Reactive Arrays:** Khi sử dụng reactive với array, Vue sẽ phản ứng với các thay đổi trong array. Ví dụ:

```
const items = reactive(['Apple', 'Banana']);  
  
items.push('Orange'); // Vue sẽ theo dõi sự thay đổi này.
```

## ❑ Reactive: Các tính năng nâng cao

- ❖ **Reactive với các thao tác trực tiếp trên Object/Array:** Có thể sử dụng các phương thức như `push`, `splice`, `delete`, v.v., và Vue sẽ cập nhật giao diện tương ứng. Ví dụ:

```
delete state.user.name; // Vue sẽ tự động cập nhật template nếu `name` được sử dụng.
```



Sử dụng kiến thức đã học thực hiện Demo sau:

Tài khoản: Nam

Vai trò: Admin

Cập nhật

Bấm Cập nhật

Tài khoản: NamDN

Vai trò: Nhân viên

Cập nhật

## ➤ Sử dụng Ref

```
<template>
  <h4>Tài khoản: {{ user.name }}</h4>
  <h4>Vai trò: {{ user.role }}</h4>

  <button class="btn btn-danger"
    @click="updateUser">Cập nhật</button>
</template>
```

Tài khoản: Nam  
Vai trò: Admin  
Cập nhật

Tài khoản: NamDN  
Vai trò: Nhân viên  
Cập nhật

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

export default {
  setup() {
    // Tạo 1 ref object user
    const user = ref({name: "Nam", role: "Admin"});

    const updateUser = () => {
      // Cập nhật ref object qua thuộc tính value
      user.value = {name: "NamDN", role: "Nhân viên"};
    };

    return {
      user, updateUser,
    };
  },
};
</script>
```



## ➤ Sử dụng Reactive

```
<template>
  <h4>Tài khoản: {{ user.name }}</h4>
  <h4>Vai trò: {{ user.role }}</h4>

  <button class="btn btn-danger"
    @click="updateUser">Cập nhật</button>
</template>
```

Tài khoản: Kien  
Vai trò: Admin  
**Cập nhật**

Tài khoản: KienDN  
Vai trò: Nhân viên  
**Cập nhật**

```
<script>
import {reactive} from "vue";

export default {
  setup() {
    // Tạo 1 reactive object user
    const user = reactive({name: "Kien", role: "Admin"});

    const updateUser = () => {
      user.name = "KienDN";
      user.role = "Nhân viên";
    };

    return {
      user, updateUser,
    };
  },
};
</script>
```

**BÀI 5:**

**DATA BINDING TRONG VUEJS**

**PHẦN II: CLASS VÀ STYLE BINDINGS**

## Class và style bindings:

Vue.js sử dụng directive **v-bind** để thực hiện bind vào HTML tag các chuỗi chứa tên class và style. Nội dung bao gồm:

### ☐ **Classes Binding (Liên kết các lớp)**

- ❖ Cú pháp kiểu Object
- ❖ Cú pháp kiểu Array

### ☐ **Binding inline styles (Ràng buộc kiểu nội tuyến)**

- ❖ Kiểu Object
- ❖ Kiểu Array

## 1. Cú pháp kiểu Object:

Trong vue.js chúng ta có thể truyền một đối tượng tới **:class** (viết tắt của **v-bind:class**) để chuyển đổi các class một cách linh động dựa trên điều kiện.

```
<div :class="{ active: isActive }"></div>
```

- ❑ Cú pháp trên có nghĩa là sự xuất hiện của lớp **active** hoàn toàn tùy thuộc vào giá trị trả về của **isActive** là **true** hay **false**. Có thể chuyển đổi nhiều class bằng cách thêm các trường vào trong đối tượng.

```
<div class="static" :class="{ active: isActive, 'text-danger': hasError }">
```

- Nếu **isActive** là **true** và **hasError** là **true** thì sẽ render ra:

```
<div class="static active text-danger"></div>
```

## 1. Cú pháp kiểu Object

```
<template>
  <div :class="{ active: isActive, 'text-danger': hasError }">Hello World</div>
</template>
<script>
import { ref } from 'vue';
export default {
  setup() {
    const isActive = ref(true);
    const hasError = ref(true);

    return { isActive, hasError };
  }
};
</script>
<!-- Class .text-danger tạo màu chữ theo định dạng của Bootstrap -->
```



The screenshot shows a web browser window with the address bar displaying 'localhost:5173'. The main content area of the browser shows the text 'Hello World' in a red font. An orange arrow points from the 'text-danger' property in the Vue.js template code to the red text in the browser, illustrating the binding between the class and the text color.

## 2. Cú pháp kiểu **Array**: Chúng ta có thể liên kết **:class** với một mảng để áp dụng danh sách các lớp

❖ Ví dụ: `<div :class="[activeClass, errorClass]"></div>`

Đồng thời khai báo:

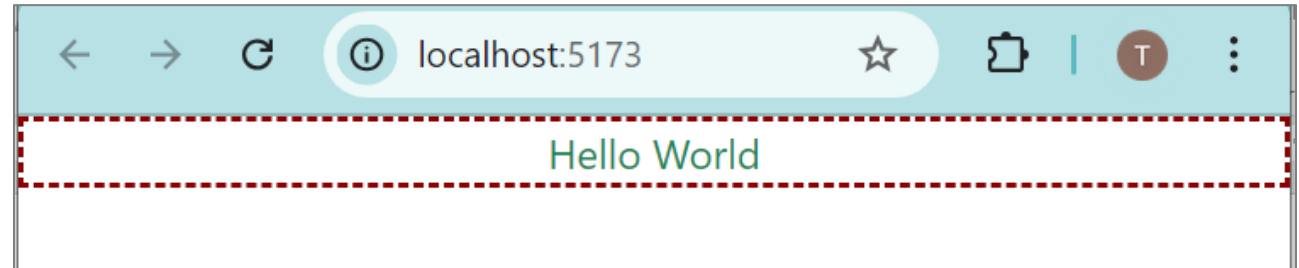
```
const activeClass = ref('active');  
const errorClass = ref('text-danger');
```

Kết quả sẽ render thành:

```
<div class="active text-danger"></div>
```

## 2. Cú pháp kiểu Array:

```
<template>
  <div :class="[isActive ? 'baseStyle' : '', 'text-success', 'text-center']">Hello World</div>
</template>
<script>
export default {
  setup() {
    const isActive = true;
    return { isActive };
  }
};
</script>
<style>
.baseStyle {
  font-size: '20px';
  border: 2px dashed darkred;
}
</style>
```



- `.text-success` và `.text-center` là 2 class được định nghĩa sẵn của Bootstrap
- `.baseStyle` là class tự định nghĩa

Tương tự class binding, Vue cũng có thể binding inline style, truyền một đối tượng tới **:style** (viết tắt của **v-bind:style**)

**1. Kiểu Object:** Sử dụng object để điều khiển các style CSS trực tiếp từ JavaScript.

Ví dụ:

```
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

Đồng thời khai báo:

```
const activeColor = ref('red')  
const fontSize = ref(30)
```

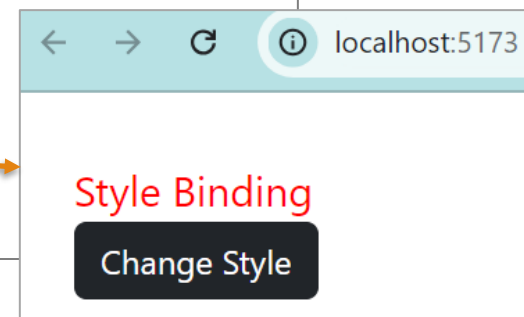
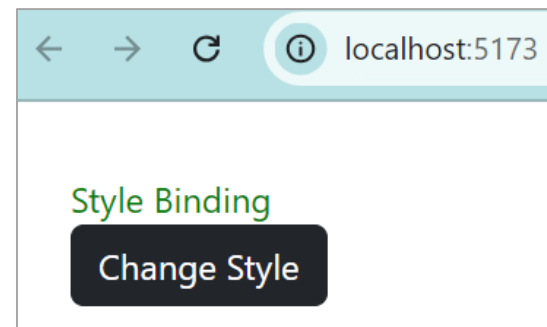


## 1. Kiểu Object:

```
<template>
  <div :style="{ color: textColor, fontSize: fontSize + 'px' }">
    Object Style Binding
  </div>
  <button class="btn btn-dark" @click="changeStyle">Change Style</button>
</template>
<script>
import { ref } from 'vue';

const textColor = ref('green');
const fontSize = ref(16);

const changeStyle = () => {
  textColor.value = textColor.value === 'green' ? 'red' : 'green';
  fontSize.value = fontSize.value === 16 ? 20 : 16;
}
</script>
```



**2. Kiểu Array:** Sử dụng **array** để điều khiển các style CSS trực tiếp từ JavaScript thông qua một mảng các object style.

❖ Cú pháp:

```
<tag :style="[style1,...,styleN]">Array Style Binding</tag>
```

❖ Trong đó:

- <tag>: tên thẻ
- style1...styleN: danh sách các style CSS điều khiển

## 2. Kiểu Array:

```
<template>
  <div :style="[style1, style2]">Array Style Binding</div>
  <button class="btn btn-info" @click="toggleStyle">Change Style</button>
</template>

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

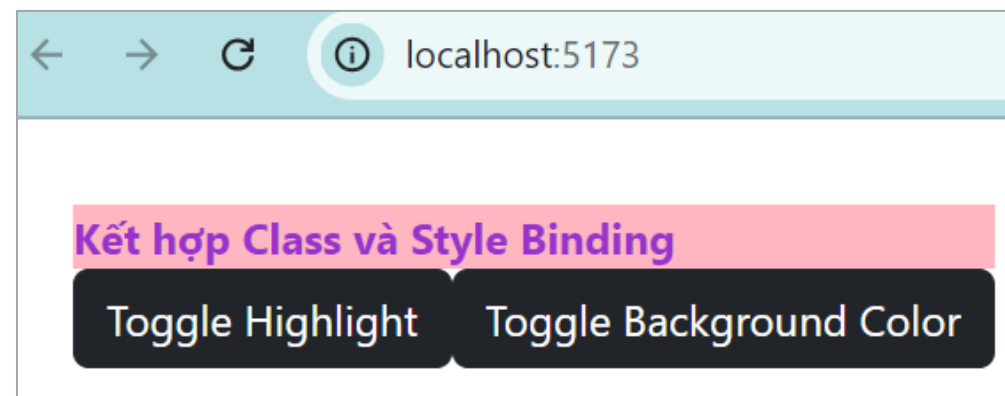
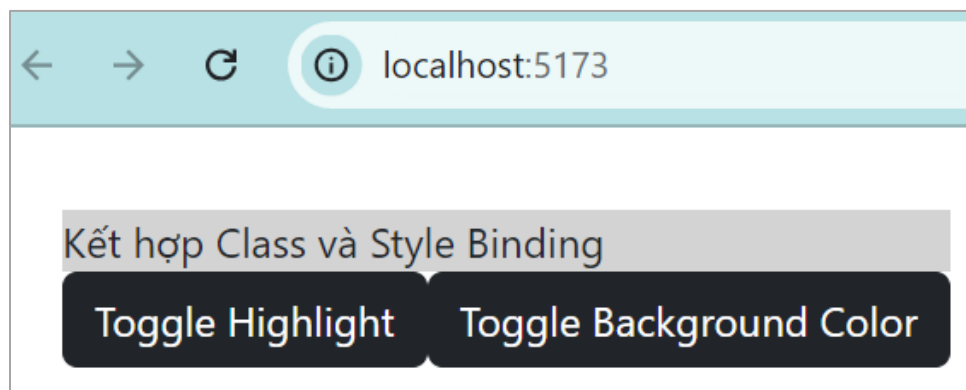
const style1 = ref({ color: 'blue' });
const style2 = ref({ fontSize: '16px' });

const toggleStyle = () => {
  style1.value = style1.value.color === 'blue' ? { color: 'red' } : { color: 'blue' };
  style2.value = style2.value.fontSize === '16px' ? { fontSize: '20px' } : { fontSize: '16px' };
}
</script>
```



# demo

**Kết Hợp Class Và Style Binding thực hiện demo sau:** Khi click vào Toogle Highlight thì đổi chữ thành màu tím, click vào Toogle Background Color thì đổi nền sang màu hồng nhạt, và ngược lại



## Hướng dẫn thực hiện demo:

```
<template>
  <div :class="{ highlighted: isHighlighted }" :style="{ backgroundColor: bgColor }">
    Kết hợp Class và Style Binding
  </div>
  <button class="btn btn-dark" @click="toggleHighlight">Toggle Highlight</button>
  <button class="btn btn-dark" @click="toggleBgColor">Toggle Background Color</button>
</template>
<script setup>
import { ref } from 'vue';
const isHighlighted = ref(false);
const bgColor = ref('lightgray');
const toggleHighlight = () => {
  isHighlighted.value = !isHighlighted.value;
}
const toggleBgColor = () => {
  bgColor.value = bgColor.value === 'lightgray' ? 'lightpink' : 'lightgray';
}
</script>
<style>.highlighted {font-weight: bold; color: darkorchid;}</style>
```

- ☑ Có hai cách bind data trong VueJs: One-way data binding (một chiều) và Two-way data binding (hai chiều).
- ☑ Có hai Reactivity API là Ref và Reactive
- ☑ Class và Style binding
  - ☑ Class Binding: Sử dụng object hoặc array để linh hoạt thêm hoặc loại bỏ các class dựa trên các điều kiện.
  - ☑ Style Binding: Sử dụng object hoặc array để điều khiển các style CSS trực tiếp từ JavaScript.



Thank  
You

