**CHƯƠNG 4: CÁC KHÁI NIỆM CƠ BẢN TRONG VUE.JS**

**4.1 Vue Instance – Tạo ứng dụng**

Trong Vue.js, **Vue Instance** (thực thể Vue) chính là điểm khởi đầu để xây dựng một ứng dụng. Mỗi ứng dụng Vue đều bắt đầu bằng việc khởi tạo một instance mới thông qua lệnh Vue.createApp().  
Instance này chịu trách nhiệm quản lý toàn bộ dữ liệu, phương thức, vòng đời và cách ứng dụng kết nối với DOM. Nó giống như “bộ não” điều khiển mọi hoạt động trong ứng dụng Vue.

Ví dụ:

*const app = Vue.createApp({*

*data() {*

*return {*

*message: "Xin chào Vue.js!"*

*}*

*}*

*});*

*app.mount('#app');*

Trong ví dụ trên, app là một Vue Instance, và nó được gắn vào phần tử DOM có id="app".

**4.2 Template – Giao diện HTML động**

Template trong Vue.js là một đoạn **HTML động**, cho phép kết hợp giữa cú pháp HTML và các biểu thức Vue. Thay vì viết thuần HTML, Vue hỗ trợ cú pháp mở rộng để hiển thị dữ liệu, xử lý sự kiện và ràng buộc thuộc tính.

Ví dụ:

*<div id="app">*

*<h1>{{ message }}</h1>*

*</div>*

Ở đây, {{ message }} là cú pháp **Interpolation**, cho phép hiển thị dữ liệu từ Vue Instance ra giao diện.

**4.3 Reactivity – Tự động cập nhật UI**

Vue nổi bật nhờ cơ chế **Reactivity** (tính phản ứng). Khi dữ liệu trong Vue Instance thay đổi, giao diện được cập nhật tự động mà không cần thao tác thủ công vào DOM.  
Điều này giúp việc phát triển trở nên dễ dàng, vì ta chỉ cần tập trung quản lý dữ liệu, còn giao diện sẽ luôn “phản chiếu” dữ liệu đó.

Ví dụ:

*data() {*

*return {*

*count: 0*

*}*

*},*

*methods: {*

*increase() {*

*this.count++;*

*}*

*}*

*<p>Bạn đã nhấn {{ count }} lần</p>*

*<button @click="increase">Nhấn tôi</button>*

Mỗi lần count thay đổi, Vue sẽ tự động cập nhật <p> mà không cần thao tác DOM thủ công.

**4.4 Data Binding – Kết nối Data & UI**

**Data Binding** là cơ chế kết nối dữ liệu trong Vue Instance với giao diện. Vue hỗ trợ hai dạng binding chính:

1. **One-way binding**: Dữ liệu từ Instance hiển thị ra giao diện.
2. <p>{{ message }}</p>
3. **Two-way binding**: Dữ liệu vừa hiển thị ra giao diện vừa được cập nhật từ giao diện.  
   Sử dụng directive v-model:
4. <input v-model="message" />
5. <p>{{ message }}</p>

Cơ chế binding giúp đồng bộ dữ liệu và UI một cách dễ dàng, giảm thiểu lỗi.

**4.5 Directives – Điều khiển DOM**

Directives trong Vue là những thuộc tính đặc biệt gắn kèm vào phần tử HTML, giúp thao tác và điều khiển DOM.

Một số directive phổ biến:

* v-if : Hiển thị hoặc ẩn phần tử theo điều kiện.
* v-for : Lặp qua mảng để render danh sách.
* v-bind : Ràng buộc giá trị thuộc tính.
* v-on : Lắng nghe sự kiện (có thể viết tắt @).

Ví dụ:

*<p v-if="isLogin">Xin chào, người dùng!</p>*

*<p v-else>Vui lòng đăng nhập</p>*

**4.6 Components – UI có thể tái sử dụng**

**Component** trong Vue là các khối giao diện độc lập, có thể tái sử dụng nhiều lần. Một ứng dụng Vue thường được chia thành nhiều component nhỏ để dễ quản lý.

Ví dụ một component:

*app.component('hello-user', {*

*template: `<h2>Xin chào {{ name }}</h2>`,*

*data() {*

*return {*

*name: 'Người dùng'*

*}*

*}*

*});*

*<hello-user></hello-user>*

*<hello-user></hello-user>*

Ở đây, hello-user có thể dùng nhiều lần mà vẫn độc lập.

**4.7 Props & Emits – Giao tiếp giữa Component**

Trong Vue, giao tiếp giữa các component thường được thực hiện thông qua **Props** và **Emits**:

* **Props**: Cho phép truyền dữ liệu từ component cha xuống component con.
* **Emits**: Cho phép component con phát ra sự kiện để cha lắng nghe.

Ví dụ:

*app.component('child-component', {*

*props: ['title'],*

*template: `*

*<div>*

*<h3>{{ title }}</h3>*

*<button @click="$emit('say-hello')">Chào</button>*

*</div>*

*`*

*});*

*<child-component title="Xin chào Vue" @say-hello="handleHello"></child-component>*

Nhờ Props và Emits, Vue hỗ trợ mô hình **cha ↔ con** rõ ràng và dễ quản lý.