TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC
—— o0o ——

THỰC TẬP KĨ THUẬT

TÌM HIỂU FRAMEWORK VUEJS

Chuyên ngành: Toán Tin

Giảng viên hướng dẫn:	**TS Lê Hải Hà**
Sinh viên thực hiện:	**Hoàng Thanh Lưu**
Mã số sinh viên:	**20162602**
Lớp:	**Toán Tin K61**

Hà Nội - 7/2020

Lời nói đầu

Trong thời đại công nghệ thông tin ngày càng phát triển nhanh và mạnh mẽ như hiện nay các phần mềm ứng dụng, trang web trực tuyến cũng không ngừng phát triển nhằm phục vụ các yêu cầu của người dùng. Xây dựng một trang web cơ bản không chỉ đáp ứng những yêu cầu, tính năng cơ bản về chức năng nghiệp vụ mà còn cần đáp ứng về mặt đẹp mắt, thân thiện, tối ưu hóa trải nghiệm người dùng. Front End là cách gọi quy trình sử dụng các ngôn ngữ HTML, CSS, JavaScript thiết kế và xây dựng giao diện cho các trang web hoặc ứng dụng web để người dùng có thể xem và tương tác trực tiếp trên đó.

Mục tiêu của việc thiết kế trang web là giúp cho người dùng dễ dàng sử dụng khi mở trang web. Điều này rất khó khăn vì trong thực tế người dụng sử dụng rất nhiều loại thiết bị khác nhau với kích thước và độ phân giải khác nhau, do đó buộc Front End Developer phải xem xét hết các khía cạnh này khi thiết kế trang web. Cần phải đảm bảo trang web xuất hiện chính xác trên các trình duyệt khác nhau, hệ điều hành khác nhau và các thiết bị khác nhau.

Trong khuôn khổ nội dung thực tập, em được giao nhiệm vụ nghiên cứu về một framework có tuổi đời khá trẻ và ngày càng được sử dụng nhiều, và hiện tại đang được sử dụng để xây dựng giao diện web tròn các dự án tại công ty Skymap, đó là VueJS.

Để có được báo cáo thực tập này em xin được gửi lời cảm ơn chân thành nhất tới Ban lãnh đạo, các phòng ban của công ty Skymap đã tạo điều kiện thuận lợi cho em được tìm hiểu, học tập, tiếp thu được những kiến thức thực tiễn trong suốt quá trình thực tập tại công ty.

Cuối cùng em xin chân thành cảm ơn các thầy cô trong Viện toán ứng dụng và tin học đã trang bị cho em những kiến thức quý báu trong suốt thời gian học tập vừa qua, qua đó em có thể vận dụng các kiến thức đã học được trong thời gian thực tập tại công ty, để rồi có thêm được nhiều kiến thức thực tế và hoàn thành tốt quá trình thực tập.

Mục lục

Chương 1

Khái quát về công ty Skymap

1.1 Giới thiệu về công ty

Công ty Trách nhiệm hữu hạn Công nghệ cao Skymap (Skymap Việt Nam) là chi nhánh của công ty Skymap Global Singapore. Công ty gồm gần 50 nhân viên chính thức trong đó TS. Lê Hải Hà hiện giữ chức vụ giám đốc công ty. Hiện tại công ty đang phát triển các lĩnh vực hoạt động chính như sau:

1.1.1 Mảng phát triển giải pháp công nghệ thông tin doanh nghiệp

Skymap có kinh nghiệm trong việc áp dụng công nghệ thông tin vào giải quyết các bài toán thực tế doanh nghiệp. Hiện công ty đang triển khai các sản phẩm hệ thống quản lý thông tin phục vụ doanh nghiệp:

- Hệ thống quản lý lao động và hỗ trợ xây dựng hồ sơ thầu
  Tên công ty: CÔNG TY CỔ PHẦN NƯỚC VÀ MÔI TRƯỜNG VIỆT NAM (VIWASE)
  Thời gian triển khai: 2 tháng

- Hệ thống quản lý lao động và tiền lương
  Tên công ty: CÔNG TY CỔ PHẦN ĐẦU TƯ THẾ GIỚI SỮA. (MILK WORLD INVESTMENT JOINT STOCK COMPANY)
  Thời gian triển khai: 3 tháng

- Sản phẩm salestrekk
  Salestrekk là hệ thống hỗ trợ bán hàng. Salestrekk là giải pháp quản lý quan hệ khách hàng, theo dõi và hỗ trợ hệ thống sale rep. Khách hàng

chính của salestrekk là các công ty bán sản phẩm theo mô hình dự án cần theo dõi quá trình chăm sóc khách hàng, các công ty có hệ thống sale, nhân viên thị trường cần quản lý và hỗ trợ tạo đơn hàng với tổng công ty.

Tính năng nổi bật của salestrekk là hệ thống bản đồ cho phép có cái nhìn trực quan về hệ thống sale, tình hình các vùng kinh doanh. Khả năng ghi lại vị trí của app di động cho phép quản lý sale và công việc chăm sóc khách hàng.

Tích hợp AI trong phân tích email khách hàng để phân tích cảm xúc khách hàng.

- Sản phẩm ứng dụng chấm công thông minh
  Thay thế máy chấm công vân tay bằng giải pháp tiết kiệm chi phí – ứng dụng chấm công trên mobile. Giải quyết được bài toán chấm công, kết nối tự động với hệ thống tính lương.

1.1.2 Mảng bản đồ

Skymap Việt Nam là công ty đứng đầu về số hóa và khai thác các hệ thống thông tin địa lý, các giải pháp toàn diện về xử lý thông tin liên quan đến dữ liệu địa lý và xử lý bản đồ.

Các dự án đang triển khai: hệ thống thông tin cháy rừng, cổng thông tin rà phá bom mìn quốc gia.

1.1.3 Mảng Machine Learning

Skymap Việt Nam hiện đang ứng dụng công nghệ machine learning và AI vào giải quyết các bài toán trích xuất thông tin từ ảnh vệ tinh. Các ứng dụng đang phát triển:

- Road detection: Nhận dạng đường từ ảnh vệ tinh.
- Building footprint: Vẽ móng các tòa nhà từ ảnh vệ tinh. Tiền đề cho phát triển bản đồ thành phố 3D.
- Tree couting: Đếm số lượng cây từ ảnh vệ tinh. Theo dõi số lượng cây trên diện tích rừng cho các công ty trồng cọ lấy dầu giúp đếm chính xác đến 95% số lượng cây cọ, phân loại cây cọ chỉ bằng ảnh máy bay không người lái (UAV).

1.2 Lịch sử phát triển

Công ty TNHH công nghệ cao Skymap (Skymap Việt Nam) được thành lập vào ngày 25/01/2017, là chi nhánh của công ty Skymap Global Singapore với ngành nghề chính là nghiên cứu và phát triển phần mềm.

Trong bối cảnh số hóa toàn cầu, xu hướng phát triển mạnh mẽ của internet, các tổ chức doanh nghiệp muốn lựa chọn những giải pháp tối ưu nhất cho việc quản lý thông tin doanh nghiệp. Chính vì đó Skymap được ra đời nằm đáp ứng các nhu cầu số hóa, quản lý thông tin doanh nghiệp với chất lượng sản phẩm tốt nhất và giá cả cạnh tranh nhất tại Hà Nội.

Trong quá trình phát triển, Skymap luôn ý thức được việc giữ gìn giá trị thương hiệu mình đã xây dựng, không ngừng hoàn thiện để khách hàng luôn đặt niềm tin vào những sản phẩm của công ty và gắn bó với Skymap như một người bạn đáng tin cậy.

1.3 Phương châm công ty

Mỗi khó khăn mà doanh nghiệp Việt đang gặp phải là một bài toán mà công ty Trách nhiệm hữu hạn công nghệ cao Skymap đang từng ngày giải quyết.

Nụ cười của doanh nghiệp là niềm vui và niềm tự hào của Skymap khi mang tới những giải pháp công nghệ tốt nhất, hoàn thiện nhất, nhanh nhất.

Chương 2

Quá trình thực tập

2.1 Các công việc được giao trong quá trình thực tập

- Tìm hiểu về framework VueJS
- Xây dựng ứng dụng nhỏ demo

2.2 Một số kiến thức cơ bản về VueJS

2.2.1 Giới thiệu về VueJS

Lịch sử hình thành và phát triển

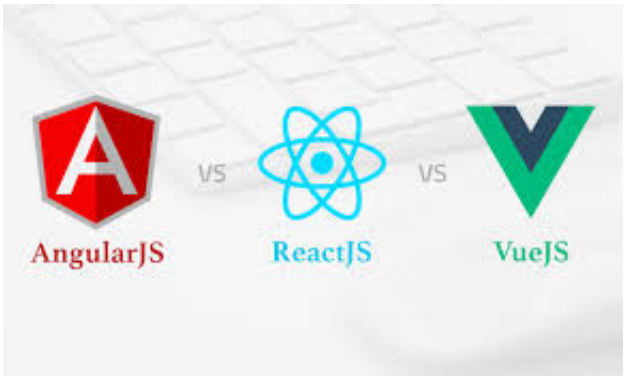
- Vue được tạo bởi Evan You sau khi làm việc ở Google, khi đó Evan đang dùng AngularJS cho một số dự án, và Evan nói rằng: "Tôi tưởng tượng, điều gì sẽ xảy ra nếu tôi trích một phần mà tôi thực sự thích về Angular và xây dựng một cái gì đó nhẹ nhàng mà không cần thêm các khái niệm bổ sung?".Vue ban đầu được phát hành lần đầu vào tháng 2 năm 2014. Dự án này đã được đăng lên HackerNew, Echo Js trong ngày đầu ra mắt.

- Hiện tại, số lượng "thích" (star) trên Github cho dự án của Vue đang ngày càng tăng nhanh. Vuejs là một trong những project phổ biến nhất trên Github và thứ 2 trong số các JavaScript Framework (chỉ sau React), Vue đã vượt qua các thư viện / framework nổi tiếng khác như Angular 1.x, JQuery, Backbonejs,...

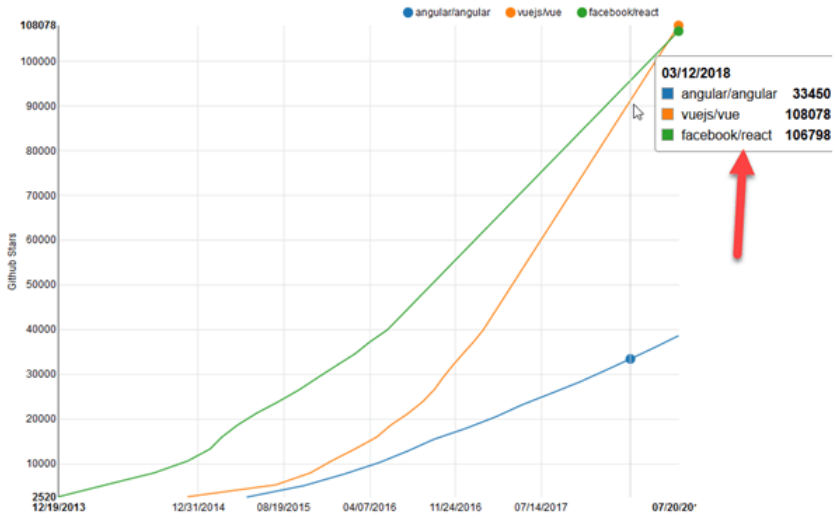
VueJS là gì?

Vue.js, gọi tắt là Vue, là một framework linh động dùng để xây dựng giao diện người dùng (user interfaces - UI). Khác với các framework nguyên khối, Vue được thiết kế từ đầu theo hướng cho phép và khuyến khích việc phát triển ứng dụng theo các bước. Khi phát triển lớp giao diện, người dùng chỉ cần dùng thư viện lõi (core library) của Vue, vốn rất dễ học và tích hợp với các thư viện hoặc dự án có sẵn. Cùng lúc đó, nếu kết hợp với những kĩ thuật hiện đại như SFC (single file components) và các thư viện hỗ trợ, Vue cũng đáp ứng được dễ dàng nhu cầu xây dựng những ứng dụng đơn trang (SPA - Single Page Applications) với độ phức tạp cao.

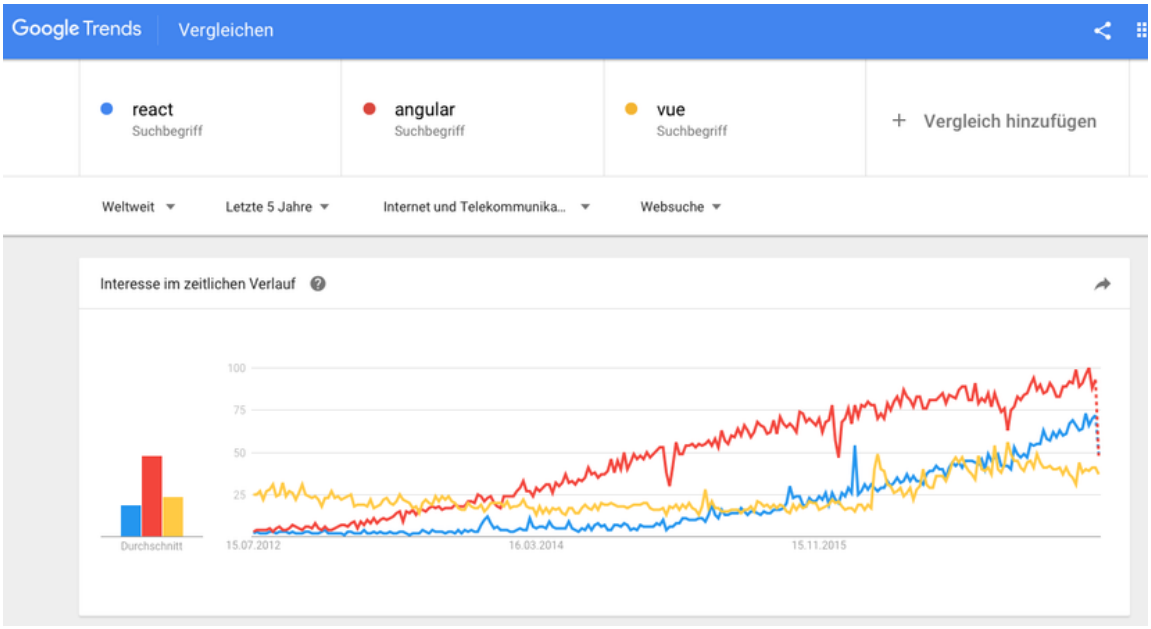
VueJS với một số framework JS khác:



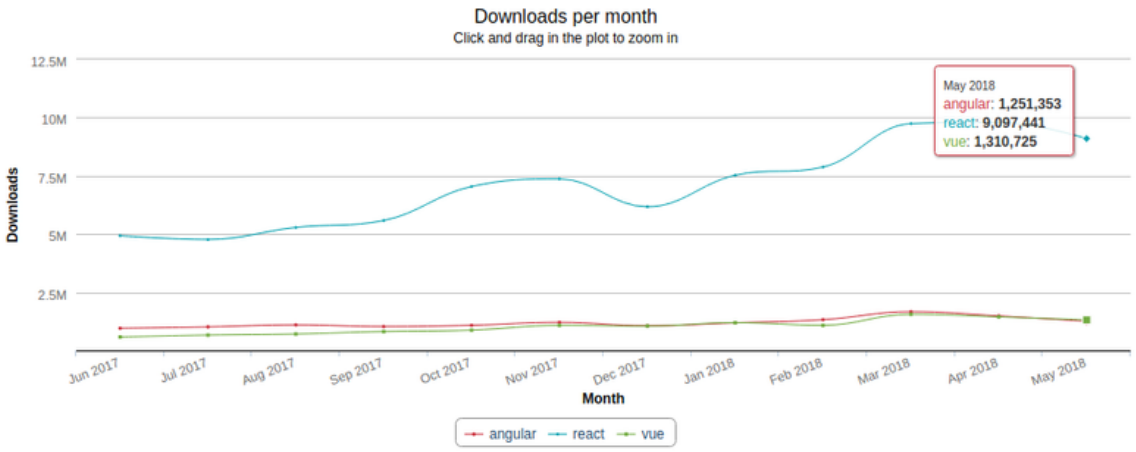
Mức độ đánh giá (Theo diễn đàn github):



Mức độ tìm hiểu:



Số lượt tải xuống:



Ưu điểm

- HTML được cấp quyền. Điều này có nghĩa rằng Vue.js có nhiều đặc điểm tương tự với Angular và điều này có thể giúp tối ưu hóa các khối HTML xử lý với việc sử dụng các components khác nhau.
- Tài liệu chi tiết. Vue.js có tài liệu hướng dẫn rất có khả năng có thể làm cho đường cong học tập trở nên nhanh hơn và tiết kiệm rất nhiều thời gian để phát triển một ứng dụng chỉ sử dụng kiến thức cơ bản về HTML và JavaScript.
- Khả năng thích ứng. Nó cung cấp một khoảng thời gian chuyển đổi nhanh từ các khung công tác khác sang Vue.js do sự tương tự với Angular và React về mặt thiết kế và kiến trúc.
- Tích hợp tuyệt vời. Vue.js có thể được sử dụng cho cả ứng dụng xây dựng một trang và giao diện web khó hơn của ứng dụng. Điều chính là các bộ phận tương tác nhỏ hơn có thể dễ dàng tích hợp vào cơ sở hạ tầng hiện tại mà không có tác động tiêu cực đến toàn bộ hệ thống.
- Large scaling (Chia tỷ lệ lớn). Vue.js có thể giúp phát triển các mẫu tái sử dụng khá lớn có thể được tạo ra mà không có thêm thời gian được phân bổ cho điều đó theo cấu trúc đơn giản của nó.
- Kích thước nhỏ. Vue.js có thể có trọng lượng khoảng 20KB giữ tốc độ và tính linh hoạt cho phép đạt hiệu suất tốt hơn nhiều so với các khung công tác khác.

Nhược điểm

- Thiếu nguồn lực. Vue.js vẫn có thị phần khá nhỏ so với React hay Angular, có nghĩa là chia sẻ tri thức trong framework này vẫn còn trong giai đoạn

đầu.

- Rủi ro về tính linh hoạt. Đôi khi, Vue.js có thể có vấn đề trong khi tích hợp vào các dự án lớn và vẫn không có kinh nghiệm với các giải pháp có thể, nhưng họ chắc chắn sẽ đến sớm.
- Thiếu tài liệu tiếng Anh đầy đủ. Điều này dẫn đến một phần phức tạp trên một số giai đoạn phát triển, tuy nhiên, ngày càng có nhiều tài liệu được dịch sang tiếng Anh.

Sử dụng

Các công ty sử dụng Vue.js: Xiaomi, Alibaba, WizzAir, EuroNews, Grammarly, Gitlab và Laracasts, Adobe, Behance, Codeship, Reuters.

2.2.2 Bắt đầu

Chỉ cần tải file thư viện về rồi sử dụng nó trong thẻ script. Vue sẽ được đăng ký thành một biến toàn cục.

Các bản: Development với đầy đủ chế độ cảnh báo, Debug, bảng Production không có các cảnh báo, nhẹ hơn, file zip.

Cách sử dụng đơn giản nhất: Copy đoạn mã nguồn sau cho vào project:

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
```

Hoặc có thể cài đặt thông qua npm: $ npm install vue

2.2.3 Đối tượng Vue

Tạo một đối tượng Vue

Một ứng dụng Vue luôn được bắt đầu bằng cách khởi tạo một đối tượng Vue (Vue instance) sử dụng hàm Vue:

```
var vm = new Vue({
  // các tùy chọn
})
```

Khi khởi tạo một đối tượng Vue, chúng ta truyền vào một object options với các tùy chọn. Một component Vue cũng là một đối tượng Vue và do đó cũng nhận cùng một object options (trừ một số tùy chọn chỉ dành riêng cho root).

Dữ liệu và phương thức

Khi một đối tượng Vue được khởi tạo, tất cả các thuộc tính (property) được tìm thấy trong object data sẽ được thêm vào reactivity system (hiểu nôm na là "hệ thống phản ứng") của Vue. Điều này có nghĩa là view sẽ "react" (phản ứng) khi giá trị của các thuộc tính này thay đổi, và tự cập nhật tương ứng với các giá trị mới.

```
// Chúng ta khởi tạo một object "data"
var data = { a: 1 }

// Object này được truyền vào một đối tượng Vue
var vm = new Vue({
  data: data
})

// Truy xuất đến thuộc tính của đối tượng
// trả về giá trị của object "data" đã khởi tạo
vm.a == data.a // => true

// Thay đổi thuộc tính của vm cũng
// ảnh hưởng đến dữ liệu ban đầu
vm.a = 2
data.a // => 2

// ... và ngược lại
data.a = 3
vm.a // => 3
```

Khi dữ liệu thay đổi, view sẽ render lại. Cũng nên lưu ý rằng một thuộc tính trong object data chỉ trở nên reactive nếu nó đã tồn tại khi chúng ta khởi tạo đối tượng Vue.

Vòng đời đối tượng

Khi được khởi tạo, một đối tượng Vue sẽ đi qua nhiều bước khác nhau - cài đặt quan sát dữ liệu (data observation), biên dịch template, gắn kết vào DOM, cập nhật DOM khi dữ liệu thay đổi v.v. Trong suốt tiến trình này, nó cũng sẽ thực thi một số hàm gọi là lifecycle hook, giúp người dùng thêm code của mình vào các giai đoạn (stage) cụ thể trong vòng đời của đối tượng.

2.2.4 Cú pháp Template

Vue.js sử dụng một cú pháp template dựa trên HTML, cho phép bạn ràng buộc (bind) một cách minh bạch cấu trúc DOM được render với dữ liệu của đối tượng Vue bên dưới. Tất cả các template của Vue đều là code HTML hợp lệ và có thể

được parse bởi các trình duyệt và parser chuẩn.

Bên dưới, Vue biên dịch template thành các hàm render Virtual DOM (DOM ảo). Kết hợp với hệ thống reactivity (phản ứng), Vue có thể xác định một cách thông minh số lượng tối thiểu các component cần phải render lại, và áp dụng số lượng tối thiểu các hiệu chỉnh về DOM khi trạng thái của ứng dụng thay đổi.

Văn bản

Hình thức ràng buộc dữ liệu cơ bản nhất là nội suy văn bản (text interpolation) sử dụng cú pháp "mustache" ("ria mép" – hai dấu ngoặc nhọn):

```
<span>Thông điệp: {{ msg }}</span>
```

Thẻ mustache sẽ được thay thế bằng giá trị của thuộc tính msg trên object data tương ứng, và cũng sẽ được cập nhật bất cứ khi nào thuộc tính này thay đổi.

HTML thuần túy

Cú pháp mustache sẽ thông dịch dữ liệu ra thành văn bản thuần túy (plain text), nghĩa là các kí tự HTML đặc biệt như <>&"' sẽ được mã hóa. Để xuất ra HTML thuần túy, bạn sẽ cần đến directive v-html.

```
<p>Sử dụng cú pháp mustache: {{ rawHtml }}</p>
<p>Sử dụng directive v-html: <span v-html="rawHtml"></span></p>
```

Các thuộc tính HTML

Cú pháp mustache không dùng được bên trong các thuộc tính HTML. Thay vào đó, bạn hãy dùng directive v-bind:

```
<div v-bind:id="dynamicId"></div>
```

Directive này cũng hoạt động với các thuộc tính boolean như disabled và selected - các thuộc tính này sẽ được bỏ đi khi biểu thức được tính toán trả về kết quả sai (falsy).

Sử dụng biểu thức Javascript

Cho đến nay chúng ta chỉ mới bind vào các khóa thuộc tính đơn giản trong template. Tuy nhiên, thật ra Vue hỗ trợ sức mạnh toàn diện của các biểu thức JavaScript bên trong toàn bộ các ràng buộc dữ liệu (data binding):

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

<div v-bind:id="'list-' + id"></div>
```

Directive

Directive là các thuộc tính đặc biệt với prefix (tiếp đầu ngữ) v-. Giá trị của thuộc tính directive phải là một biểu thức JavaScript đơn lẻ (ngoại trừ v-for mà chúng ta sẽ đề cập sau). Nhiệm vụ của một directive là áp dụng các hiệu ứng phụ vào DOM khi giá trị của biểu thức thay đổi.

Tham số

Một số directive có thể nhận vào một tham số, đánh dấu bằng một dấu hai chấm theo sau tên của directive. Ví dụ, directive v-bind được dùng để cập nhật động một thuộc tính HTML:

```
<a v-bind:href="url"> ... </a>
```

Ở đây href là tham số hướng dẫn directive v-bind ràng buộc thuộc tính href vào giá trị của biểu thức url.

Modifier

Modifier là các hậu tố (postfix) đặc biệt được đánh dấu bằng một dấu chấm, chỉ rõ rằng một directive phải được ràng buộc theo một cách đặc biệt nào đó. Ví dụ, modifier .prevent hướng dẫn directive v-on gọi event.preventDefault() khi sự kiện được kích hoạt:

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

Cú pháp rút gọn

Prefix v- đóng vai trò gợi ý trực quan để nhận ra các thuộc tính riêng của Vue trong template. Điều này có ích khi bạn sử dụng Vue vào các dự án có sẵn, tuy nhiên đối với các directive được dùng thường xuyên thì v- có thể trông hơi rườm rà. Thêm vào đó v- trở nên kém quan trọng hơn khi bạn xây dựng các ứng dụng một trang, trong đó Vue quản lí toàn bộ các template. Vì thế Vue cung cấp dạng rút gọn (shorthand) đặc biệt cho hai trong số các directive được dùng

nhiều nhất, v-bind và v-on:

v-bind

```
<!-- cú pháp đầy đủ -->
<a v-bind:href="url"> ... </a>

<!-- cú pháp rút gọn: dùng dấu hai chấm -->
<a :href="url"> ... </a>
```

v-on

```
<!-- cú pháp đầy đủ -->
<a v-on:click="doSomething"> ... </a>

<!-- cú pháp rút gọn: dùng kí tự @ -->
<a @click="doSomething"> ... </a>
```

2.2.5 Computed property và watcher

Computed property

Viết biểu thức trực tiếp trong template rất tiện, nhưng chỉ dành cho những biểu thức có tính toán đơn giản. Những biểu thức phức tạp được viết theo cách đó sẽ khiến template cồng kềnh và khó bảo trì. Đó là lí do tại sao đối với bất kì logic nào phức tạp, bạn nên sử dụng computed property. Ví dụ:

HTML
```
<div id="example">
  <p>Thông điệp ban đầu: "{{ message }}"</p>
  <p>Thông điệp bị đảo ngược bằng tính toán (computed): "{{ reversedMessage }}"</p>
</div>
```

JS
```
var vm = new Vue({
  el: '#example',
  data: {
    message: 'người đông bến đợi thuyền xuôi ngược'
  },
  computed: {
    // một computed getter
    reversedMessage: function () {
      // `this` trỏ tới đối tượng vm
      return this.message.split(' ').reverse().join(' ')
    }
  }
})
```

Kết quả là:

Thông điệp ban đầu: "người đông bến đợi thuyền xuôi ngược"
Thông điệp bị đảo ngược (computed): "ngược xuôi thuyền đợi bến đông người"

Bạn có thể ràng buộc dữ liệu (data-bind) cho computed property trong template một cách bình thường như những thuộc tính khác. Vue biết được vm.reversedMessage phụ thuộc vào vm.message nên sẽ cập nhật bất kì ràng buộc (binding) nào phụ thuộc vào vm.reversedMessage khi vm.message thay đổi. Điểm hay nhất ở đây là chúng ta tạo ra được mối liên hệ giữa các thành phần phụ thuộc (dependency): các hàm getter của computed thì không bị hiệu ứng phụ (side effect), chính điều đó giúp dễ hiểu và dễ kiểm tra.

Computed caching và phương thức

Bạn có lẽ đã nhận ra chúng ta cũng có thể đạt được cùng một kết quả bằng cách sử dụng một phương thức:

```
<p>Thông điệp bị đảo ngược: "{{ reverseMessage() }}"</p>
```

```
// trong component
methods: {
  reverseMessage: function () {
    return this.message.split(' ').reverse().join(' ')
  }
}
```

Thay vì sử dụng computed property, chúng ta cũng có thể dùng một phương thức thay thế. Nếu xét về kết quả cuối cùng thì hai cách tiếp cận này thât ra chỉ là một. Tuy nhiên, sự khác biệt ở đây là computed property được cache lại dựa vào những những thành phần phụ thuộc (dependency). Một computed property chỉ được tính toán lại khi những thành phần phụ thuộc của chúng thay đổi. Điều này có nghĩa: miễn là giá trị của message không thay đổi, thì những truy cập tới computed reversedMessage sẽ ngay lập tức trả về kết quả được tính toán trước đó mà không phải chạy lại hàm một lần nữa.

Để so sánh, một phương phương thức luôn được gọi khi có một sự kiện render lại (re-render) xảy ra.

Tại sao chúng ta lại cần phải cache? Thử tưởng tượng chúng ta có một computed property A có nhiều thao tác tính toán trên một mảng dữ liệu lớn. Chúng ta lại có nhiều computed property phụ thuộc vào A. Nếu không cache lại, chúng

ta phải thực thi hàm getter của A nhiều hơn mức cần thiết rất nhiều! Trong trường hợp bạn không muốn cache, hãy sử dụng một phương thức thay thế.

Watcher

Computed property thích hợp cho hầu hết các trường hợp, nhưng cũng có lúc cần tới những watcher tùy biến. Đó là lí do tại sao Vue cung cấp một cách khái quát hơn để phản ứng lại với việc thay đổi dữ liệu trong watch. Cách sử dụng này rất hữu ích khi bạn muốn thực hiện những tính toán không đồng bộ và tốn kém liên quan đến việc thay đổi dữ liệu.

Ngoài tùy chọn watch, bạn cũng có thể sử dụng vm.$watch API.

2.2.6 Binding cho class và style

Binding cho class trong HTML

Sử dụng cú pháp Object:
Ta có thể truyền một object vào v-bind:class để bật tắt class một cách linh hoạt:

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

Cú pháp như trên nghĩa là class active sẽ được áp dụng tùy theo tính đúng sai (truthiness) của thuộc tính dữ liệu isActive.

Bạn có thể bật tắt nhiều class bằng cách dùng nhiều field (trường) trong object. Thêm vào đó, directive v-bind:class và thuộc tính class thông thường có thể được dùng cùng lúc với nhau. Nếu chúng ta có template sau:

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

và dữ liệu truyền vào như thế này

```
data: {
  isActive: true,
  hasError: false
}
```

thì kết quả render sẽ là:

```
<div class="static active"></div>
```

Khi giá trị isActive hoặc hasError thay đổi, danh sách class sẽ được cập nhật tương ứng. Ví dụ, nếu hasError trở thành true, danh sách class sẽ trở thành "static active text-danger".

Object được bind cũng không bắt buộc phải khai báo trong template:

```
<div v-bind:class="classObject"></div>
```

```
data: {
  classObject: {
    active: true,
    'text-danger': false
  }
}
```

Ví dụ trên sẽ render ra cùng một kết quả. Chúng ta cũng có thể bind vào một computed property (thuộc tính được tính toán) trả về một object. Dưới đây là một ví dụ điển hình cho kĩ thuật này:

```
<div v-bind:class="classObject"></div>
```

```
data: {
  isActive: true,
  error: null
},

computed: {
  classObject: function () {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type === 'fatal'
    }
  }
}
```

Sử dụng cú pháp mảng:

Chúng ta có thể truyền một mảng vào v-bind:class để áp dụng một danh sách class:

```
<div v-bind:class="[activeClass, errorClass]"></div>
```

```
data: {
  activeClass: 'active',
  errorClass: 'text-danger'
}
```

sẽ render ra kết quả sau:

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

Nếu muốn bật tắt theo điều kiện một class trong danh sách, bạn có thể dùng một toán tử ba ngôi (ternary expression):

```
<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>
```

Đoạn code trên sẽ luôn luôn áp dụng class errorClass, nhưng chỉ áp dụng activeClass khi isActive mang giá trị đúng.

Cách làm này có thể hơi dài dòng nếu bạn có nhiều class theo điều kiện. Do đó, bạn có thể dùng cú pháp object bên trong cú pháp mảng, như sau:

```
<div v-bind:class="[{ active: isActive }, errorClass]"></div>
```

Binding cho inline style

Sử dụng cú pháp object:

Cú pháp object của v-bind:style rất đơn giản - trông giống như CSS thông thường, chỉ khác ở chỗ nó là một object JavaScript. Bạn có thể dùng camelCase hoặc kebab-case (đặt trong dấu nháy nếu là kebab-case) đối với tên thuộc tính CSS:

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

```
data: {
  activeColor: 'red',
  fontSize: 30
}
```

Thông thường thì ta nên bind vào một object dành riêng cho style để template được gọn gàng hơn:

```
<div v-bind:style="styleObject"></div>
```

```
data: {
  styleObject: {
    color: 'red',
    fontSize: '13px'
  }
}
```

Một lần nữa, cú pháp object thường được dùng kết hợp với các computed property trả về object.

Sử dụng cú pháp mảng:

Cú pháp mảng của v-bind:style giúp bạn áp dụng nhiều object style cho cùng một phần tử web:

```
<div v-bind:style="[baseStyles, overridingStyles]"></div>
```

Tự động theeo Prefix:

Khi bạn sử dụng một thuộc tính CSS còn khá mới và cần vendor prefix trong v-bind:style, ví dụ transform, Vue sẽ tự động phát hiện và thêm prefix thích hợp vào các style được áp dụng.

Nhiều giá trị:

Có thể cung cấp một mảng các giá trị (đã có prefix) cho một thuộc tính CSS, như sau:

```
<div v-bind:style="{ display: ['-webkit-box', '-ms-flexbox', 'flex'] }"></div>
```

Với kĩ thuật này, Vue sẽ chỉ render ra giá trị cuối cùng trong mảng mà trình duyệt hỗ trợ. Trong ví dụ trên, Vue sẽ render display: flex trên các trình duyệt hỗ trợ flexbox.

2.2.7 Render theo điều kiện

v-if

Trong các thư viện biên dịch template như Handlebars, thông thường chúng ta sẽ viết các khối điều kiện (conditional block) như sau:

```
{{#if ok}}
  <h1>Mọi việc ổn cả</h1>
{{/if}}
```

Để làm điều này với Vue, chúng ta sử dụng directive v-if:

```
<h1 v-if="ok">Mọi việc ổn cả</h1>
```

Chúng ta cũng có thể thêm khối "else" bằng v-else:

```
<h1 v-if="ok">Mọi việc ổn cả</h1>
<h1 v-else>Có gì đó sai sai</h1>
```

Nhóm điều kiện với v-if trên thẻ <template>: Vì là một directive, v-if phải được dùng trên một phần tử đơn lẻ (single element) như <p> hoặc . Nếu chúng ta muốn bật tắt một nhóm các phần tử thì sao? Chỉ cần dùng v-if

trên một phần tử <template> với vai trò wrap (bọc) các phần tử lại thành một nhóm. Kết quả render cuối cùng sẽ không có phần tử <template> này.

```
<template v-if="ok">
  <h1>Anh chàng chăn lợn</h1>
  <p>Ơ này, Augustin thân mến ơi</p>
  <p>Mọi việc đều như ý, như ý, như ý</p>
</template>
```

Directive v-else-if đóng vai trò một khối "else if" cho v-if. Directive này có thể được dùng nhiều lần nối tiếp nhau:

```
<div v-if="type === 'A'">A</div>
<div v-else-if="type === 'B'">B</div>
<div v-else-if="type === 'C'">C</div>
<div v-else-if="type === 'D'">D</div>
<div v-else>Chiến Sĩ không thể nhớ nổi</div>
```

Tương tự với v-else, phần tử với v-else-if phải theo ngay sau một phần tử v-if hoặc v-else-if.

v-show

Một lựa chọn nữa cho việc hiện hoặc ẩn một phần tử web theo điều kiện là directive v-show. Cách dùng v-show cũng tương tự với v-if:

```
<h1 v-show="ok">Xin chào!</h1>
```

Điểm khác biệt giữa v-show và v-if là phần tử được đánh dấu với v-show sẽ luôn luôn được render và chứa trong DOM; v-show chỉ bật tắt thuộc tính display của phần tử này.

2.2.8 Render theo danh sách

Map một mảng thành các phần tử web với v-for

Chúng ta có thể dùng directive v-for để render một danh sách các item dựa trên một mảng. Directive v-for đòi hỏi một cú pháp đặc biệt dưới dạng item in items, trong đó items là mảng dữ liệu nguồn và item trỏ đến phần tử mảng đang được duyệt đến:

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.name }}
  </li>
</ul>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { name: 'Cà phê' },
      { name: 'Trà đặc' },
      { name: 'Bò húc' }
    ]
  }
})
```

Kết quả là

- Cà phê
- Trà đặc
- Bò húc

Bên trong vòng lặp v-for chúng ta có toàn quyền truy xuất đến các thuộc tính của scope cha. v-for cũng hỗ trợ một tham số thứ hai (không bắt buộc) chỉ số thứ tự (index) của phần tử mảng hiện hành.

```
<ul id="example-2">
  <li v-for="(item, index) in items">
    {{ parentMessage }} - {{ index }} - {{ item.name }}
  </li>
</ul>
```

```
var example2 = new Vue({
  el: '#example-2',
  data: {
    parentMessage: 'Parent',
    items: [
      { name: 'Cà phê' },
      { name: 'Trà đặc' },
      { name: 'Bò húc' }
    ]
  }
})
```

Kết quả là:

- Parent - 0 - Cà phê
- Parent - 1 - Trà đặc
- Parent - 2 - Bò húc

Bạn cũng có thể dùng of để phân cách thay vì in. Cách này cũng gần hơn với cú pháp vòng lặp trong JavaScript.

Dùng v-for với một object

Bạn cũng có thể dùng v-for để duyệt qua các thuộc tính của một object.

```
<ul id="v-for-object" class="demo">
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
```

```
new Vue({
  el: '#v-for-object',
  data: {
    object: {
      // tất nhiên chúng ta đều biết ông Bành Tố không phải
      // họ Bành tên Tố, nhưng đây chỉ là một ví dụ…
      'họ': 'Bành',
      'tên': 'Tố',
      'tuổi': 800
    }
  }
})
```

Kết quả

- Bành
- Tổ
- 800

Bạn cũng có thể cung cấp tham số thứ hai dùng cho khóa (key) của thuộc tính:

```
<div v-for="(value, key) in object">
  {{ key }}: {{ value }}
</div>
```

họ: Bành
tên: Tổ
tuổi: 800

Hiển thị kết quả đã được lọc hoặc sắp xếp

Đôi khi chúng ta muốn hiển thị một phiên bản đã được lọc (filter) hoặc sắp xếp (sort) của một mảng mà không thay đổi mảng đó. Trong trường hợp này, bạn có thể tạo một computed property trả về mảng đã được lọc hoặc sắp xếp. Ví dụ:

```
<li v-for="n in evenNumbers">{{ n }}</li>
```

```
data: {
  numbers: [ 1, 2, 3, 4, 5 ]
},

computed: {
  evenNumbers: function () {
    return this.numbers.filter(function (number) {
      return number % 2 === 0
    })
  }
}
```

Trong trường hợp không dùng được computed property (ví dụ trong các vòng lặp v-for), ta có thể dùng một phương thức:

```
<li v-for="n in even(numbers)">{{ n }}</li>
```

```
data: {
  numbers: [ 1, 2, 3, 4, 5 ]
},

methods: {
  even: function (numbers) {
    return numbers.filter(function (number) {
      return number % 2 === 0
    })
  }
}
```

v-for dùng với một dãy

v-for cũng có thể nhận một số nguyên n. Trong trường hợp này, template sẽ được lặp lại n lần:

```
<div>
  <span v-for="n in 10">{{ n }} </span>
</div>
```

Kết quả là: 1 2 3 4 5 6 7 8 9 10.

v-for dùng với thẻ <template>

Tương tự với v-if, bạn có thể dùng v-for trên một thẻ <template> để render một lúc nhiều phần tử. Ví dụ:

```
<ul>
  <template v-for="item in items">
    <li>{{ item.msg }}</li>
    <li class="divider"></li>
  </template>
</ul>
```

v-for dùng với v-if

Khi được dùng trên dùng một node, v-for có độ ưu tiên cao hơn v-if, có nghĩa là v-if sẽ được thực thi một cách riêng biệt trên mỗi vòng lặp của v-for. Điều này có thể có ích khi bạn muốn render cho chỉ một số item, như trong ví dụ sau:

```
<li v-for="todo in todos" v-if="!todo.isComplete">
  {{ todo }}
</li>
```

Ví dụ trên sẽ chỉ render những todo chưa hoàn thành.

Ngược lại, nếu bạn muốn bỏ qua việc thực thi vòng lặp v-for theo điều kiện, hãy dùng v-if trên một phần tử wrapper (hoặc <template>). Ví dụ:

```
<ul v-if="todos.length">
  <li v-for="todo in todos">
    {{ todo }}
  </li>
</ul>
<p v-else>Mọi việc đã hoàn thành.</p>
```

2.2.9 Xử lí sự kiện

Lắng nghe sự kiện

Chúng ta có thể dùng directive v-on để lắng nghe các sự kiện DOM và thực thi JavaScript khi những sự kiện này được kích hoạt. Ví dụ:

```
<div id="example-1">
  <button v-on:click="counter += 1">Đếm cừu</button>
  <p>{{ counter }} con cừu.</p>
</div>
```

```
var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

Phương thức xử lí sự kiện

Trong thực tế, logic để xử lí sự kiện thường phức tạp hơn, vì thế chứa JavaScript trực tiếp trong giá trị của thuộc tính v-on như trên là không khả thi. Đó là lí do v-on cũng có thể nhận tên của một phương thức mà bạn muốn gọi. Ví dụ:

```
<div id="example-2">
  <!-- `greet` là tên của phương thức bạn muốn gọi -->
  <button v-on:click="greet">Chào mừng</button>
</div>
```

```
var example2 = new Vue({
  el: '#example-2',
  data: {
    name: 'Vue.js'
  },
  // định nghĩa phương thức trong object `methods`
  methods: {
    greet: function (event) {
      // bên trong một phương thức, `this` trỏ đến đối tượng Vue
      alert('Xin chào ' + this.name + '!')
      // `event` là sự kiện DOM native
      if (event) {
        alert(event.target.tagName)
      }
    }
  }
})

// bạn cũng có thể gọi phương thức từ JavaScript
example2.greet() // => 'Xin chào Vue.js!'
```

Gọi phương thức inline

Thay vì bind trực tiếp tên phương thức, ta cũng có thể gọi phương thức trong một câu lệnh JavaScript:

```
<div id="example-3">
  <button v-on:click="say('Konica')">Hãy nói Konica</button>
  <button v-on:click="say('Viettel')">Hãy nói theo cách của bạn</button>
</div>
```

```
new Vue({
  el: '#example-3',
  methods: {
    say: function (message) {
      alert(message)
    }
  }
})
```

Đôi khi chúng ta cũng muốn truy xuất đến sự kiện DOM ban đầu từ câu lệnh JavaScript inline. Bạn có thể truyền sự kiện DOM vào phương thức thông qua biến $event:

```
<button v-on:click="warn('Chưa submit được form.', $event)">
  Submit
</button>
```

```
// ...
methods: {
  warn: function (message, event) {
    // bây giờ chúng ta có thể truy xuất đến sự kiện DOM native
    if (event) event.preventDefault()
    alert(message)
  }
}
```

Event modifier

Trong rất nhiều trường hợp, chúng ta cần gọi event.preventDefault() hoặc là gọi event.stopPropagation() bên trong một phương thức xử lí sự kiện. Tuy việc này không có gì khó, sẽ tốt hơn nếu các phương thức chỉ phải tập trung giải quyết logic dữ liệu thay vì cáng đáng các sự kiện DOM. Để giải quyết vấn đề này, Vue cung cấp các event modifier cho v-on. Event modfier là một hậu tố (postfix) cho directive, được biểu thị bằng một dấu chấm.
".stop .prevent .capture .self .once"

Key modifier

Khi lắng nghe các sự kiện bàn phím (keyboard event), chúng ta thường phải kiểm tra mã phím (key code). Vue hỗ trợ thêm key modifier (modifer cho mã phím) cho v-on trong các trường hợp này.

Tại sao lại lắng nghe sự kiện trong HTML?

Bạn có thể lo ngại rằng toàn bộ việc lắng nghe sự kiện bằng cách đặt event listener trong HTML như thế này là vi phạm quy tắc "separation of concerns." Cứ yên tâm, vì tất cả các hàm và biểu thức xử lí sự kiện của Vue được ràng buộc chặt chẽ với ViewModel, sẽ không có khó khăn gì trong việc bảo trì. Thật ra, sử dụng v-on còn có những lợi ích sau:

1. Giúp định vị hàm xử lí trong code JavaScript được dễ dàng hơn bằng cách đọc lướt template HTML.

2. Vì không phải attach hàm xử lí sự kiện trong JavaScript một cách thủ công, code trong ViewModel trở nên thuần logic và không phụ thuộc vào DOM. Điều này giúp chúng ta dễ viết test.

3. Khi một ViewModel bị hủy, tất cả hàm xử lí sự kiện đính kèm cũng được tự động gỡ bỏ mà không cần bạn phải dọn dẹp.

2.2.10 Cơ bản về Component

Ví dụ cơ bản

Đây là ví dụ về một component trong Vue:

```
// Định nghĩa một component với tên là "button-counter"
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">Bạn đã bấm {{ count }} lần.</button>'
})
```

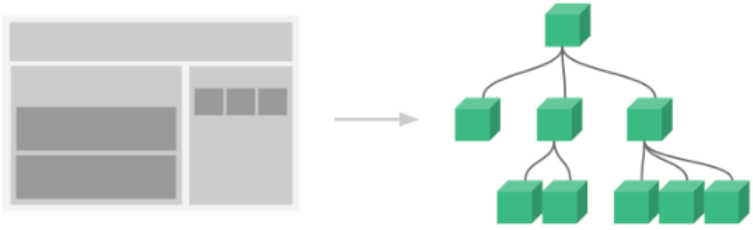
Component là các đối tượng Vue có thể sử dụng lại được với một cái tên: trong trường hợp này là <button-counter>. Chúng ta có thể dùng component này như là một phần tử bên trong đối tượng Vue gốc được tạo bởi new Vue:

```
<div id="components-demo">
  <button-counter></button-counter>
</div>
```

```
new Vue({ el: '#components-demo' })
```

Tổ chức component

Một ứng dụng thường được tổ chức dưới dạng một cây component lồng nhau:



Để có thể được sử dụng trong các template, component phải được đăng kí. Có hai cách đăng kí component: toàn cục và cục bộ. Trên đây chúng ta chỉ mới đăng kí component ở cấp toàn cục với Vue.component:

```
Vue.component('my-component-name', {
  // ... tùy chọn ...
})
```

Component đăng kí ở cấp toàn cục có thể được dùng trong template của bất kì đối tượng Vue gốc (new Vue) nào được tạo ra sau đó – và trong tất cả các component con trên cây component của đối tượng đó.

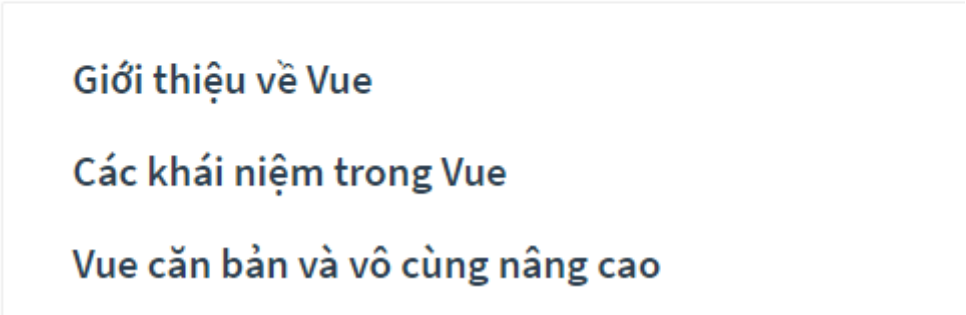
Truyền dữ liệu xuống component con bằng prop

Prop là các thuộc tính tùy chỉnh mà bạn có thể đăng kí trên một component. Khi một giá trị được truyền vào một prop, nó trở thành một "_prop_erty" của đối tượng component đó. Để truyền tựa đề (title) vào component bài viết (blog-post), chúng ta sử dụng tùy chọn props:

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

Một component có thể có bao nhiêu prop tùy ý, và prop có thể nhận bất kì giá trị gì. Trong template trên đây, bạn có thể thấy là chúng ta có thể truy xuất giá trị này trên đối tượng component, giống như với data. Một khi prop đã được đăng kí, bạn có thể truyền dữ liệu vào như một thuộc tính tùy chỉnh, ví dụ:

```
<blog-post title="Giới thiệu về Vue"></blog-post>
<blog-post title="Các khái niệm trong Vue"></blog-post>
<blog-post title="Vue căn bản và vô cùng nâng cao"></blog-post>
```



Tuy nhiên, trong một ứng dụng điển hình, bạn có lẽ sẽ có một mảng các bài viết trong data:

```
new Vue({
  el: '#blog-post-demo',
  data: {
    posts: [
      { id: 1, title: 'Giới thiệu về Vue' },
      { id: 2, title: 'Các khái niệm trong Vue' },
      { id: 3, title: 'Vue căn bản và vô cùng nâng cao' }
    ]
  }
})
```

và sau đó render một component cho mỗi bài viết:

```
<blog-post
  v-for="post in posts"
  v-bind:key="post.id"
  v-bind:title="post.title"
></blog-post>
```

Trên đây, bạn có thể thấy là chúng ta dùng v-bind để truyền động prop. Cách làm này đặc biệt hữu ích khi bạn không biết trước được chính xác nội dung bạn sẽ render, như khi lấy bài viết từ một API.

Một phần tử gốc đơn lập

Khi xây dựng component <blog-post> cho bài viết, thế nào rồi template của bạn cũng sẽ chứa nhiều thứ hơn là mỗi title. Ít nhất bạn cũng sẽ có thêm nội dung bài viết:

```
<div class="blog-post">
  <h3>{{ post.title }}</h3>
  <div v-html="post.content"></div>
</div>
```

Nhưng nếu bạn sử dụng template này, Vue sẽ thông báo lỗi every component must have a single root element (mỗi component phải có một phần tử gốc đơn lập). Bạn có thể sửa lỗi này bằng cách bọc template trong một phần tử cha, ví dụ:

```
<div class="blog-post">
  <h3>{{ post.title }}</h3>
  <div v-html="post.content"></div>
</div>
```

Component động

Đôi khi bạn muốn chuyển qua lại giữa các component, ví dụ như trên một giao diện tab:

Ví dụ trên hoạt động nhờ thuộc tính đặc biệt is của một component trong Vue:

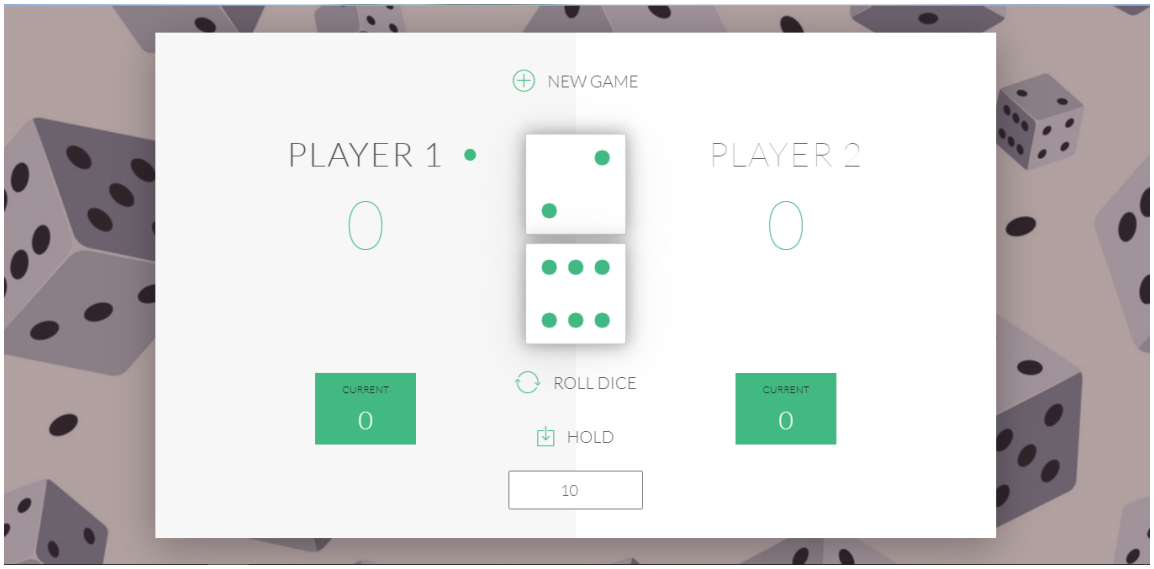
```
<!-- Component thay đổi khi currentTabComponent thay đổi -->
<component v-bind:is="currentTabComponent"></component>
```

Trong ví dụ trên, currentTabComponent có thể chứa:

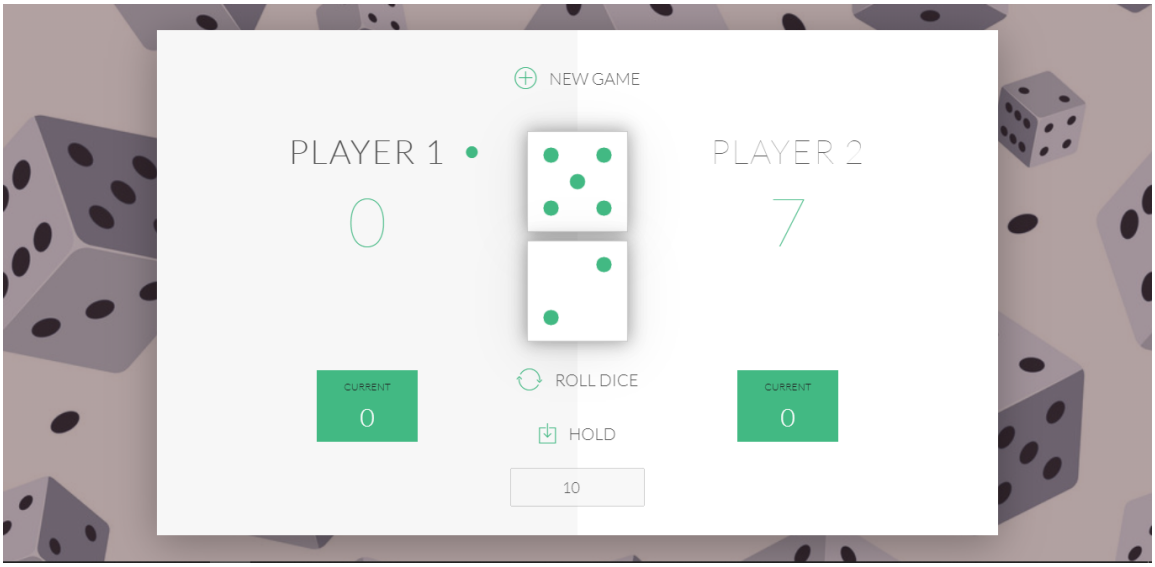
- tên của một component đã được đăng kí, hoặc
- object chứa các tùy chọn của một component

2.3 Xây dựng game xúc xắc sử dụng VueJS

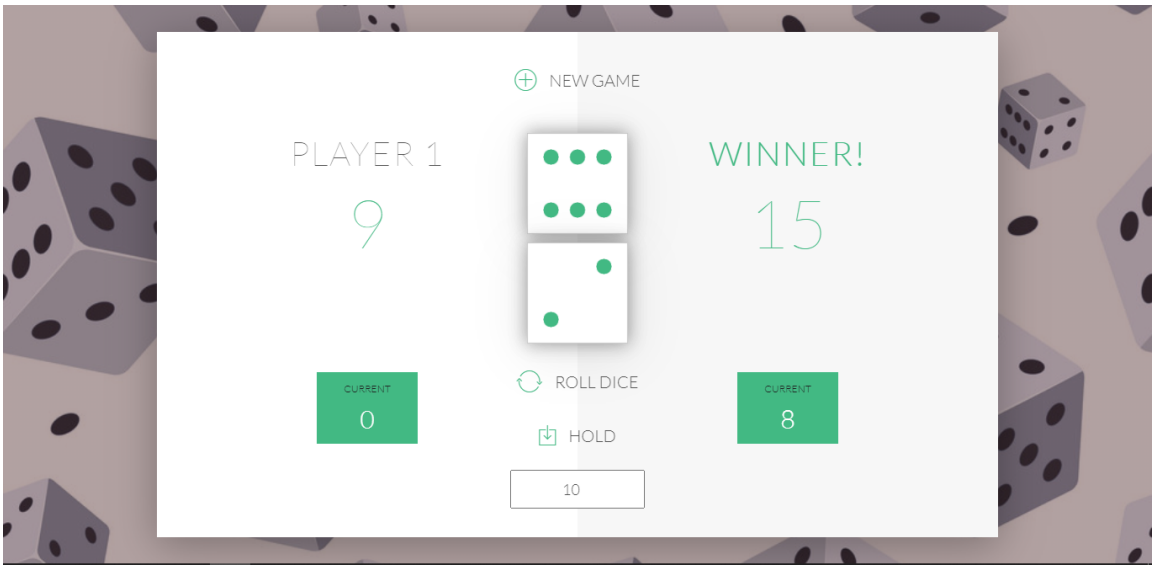
Giao diện trang chủ



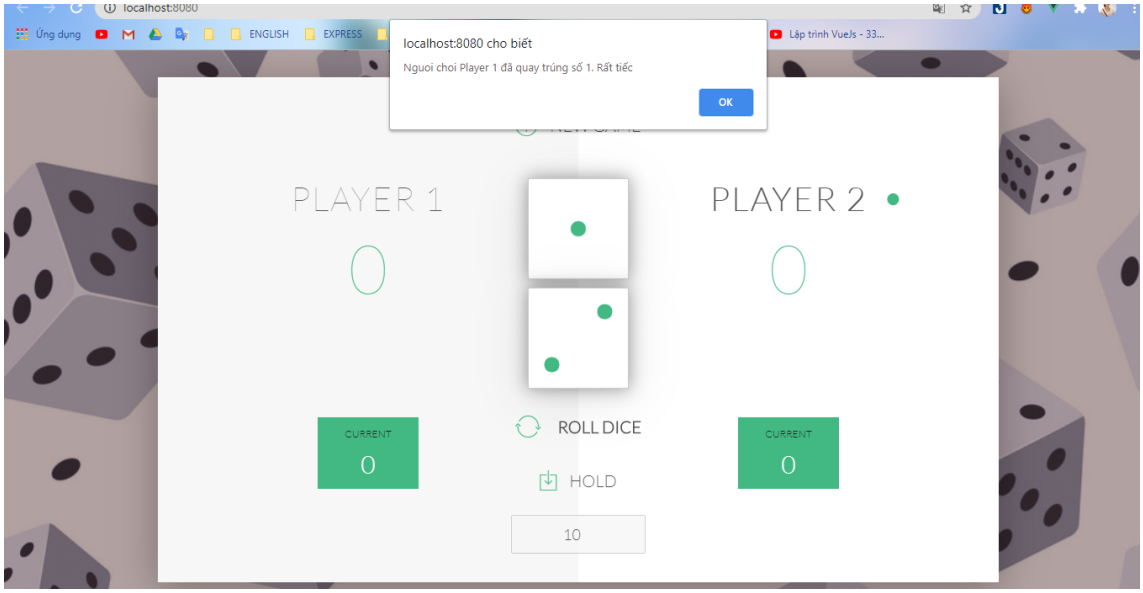
Giao diện người chơi



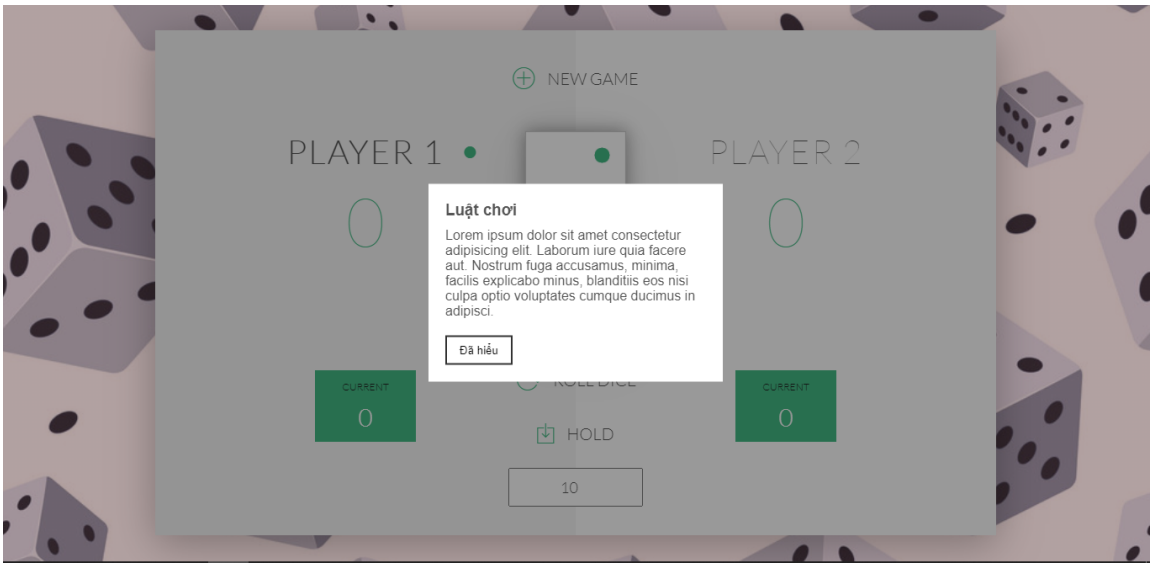
Giao diện người chiến thắng



Giao diện người thua cuộc



Giao diện luật chơi



Chương 3

Kết luận

Trên đây là những kiến thức em tìm hiểu cũng như được học trong suốt thời gian thực tập tại công ty. Sau gần 10 tuần thực tập tại công ty em đã có thêm được nhiều kĩ năng và kiến thức nhất định về lập trình web, lập trình front-end sử dụng framework VueJS. Em cũng đã xây dựng được một ứng dụng nhỏ demo để vận dụng các kiến thức tìm hiểu được từ VueJS.

Do thời gian tìm hiểu cũng như kiến thức còn hạn chế nên bài báo cáo của em không tránh khỏi thiếu sót, em rất mong có được sự đóng góp chỉnh sửa từ các thầy cô và anh chị để bài báo cáo của em được hoàn thiện hơn.

Em xin chân thành cảm ơn.

Tài liệu tham khảo

[1] https://vi.wikipedia.org/wiki/Vue.js

[2] https://vuejs.org/

[3] https://viblo.asia/p/so-sanh-giua-reactjs-va-vuejs-va-angular-3Q75wdrQKWb