Handle Media Queries and Responsive Design with Vue.js

alligator.io/vuejs/vue-media-queries

Responsive design is now a prerequisite to most web applications. As developers, we have to deal with a variety of devices and screen sizes. CSS is a great tool for simple use-cases and very efficient performance-wise. However, it was designed for documents rather than applications and it makes it painful to setup complex behaviors, even with the use of preprocessors (SASS, PostCSS, LESS,...).

Thanks to the MatchMedia API, Vue.js can greatly reduce the complexity of handling media queries and responsive design. It provides a seamless integration with a component-based architecture, keeping a clean declarative and semantic approach.

Let's see how you can do it using the `vue-mq` plugin.

This plugin relies on the matchMedia API to detect screen size changes. So, for older browsers and IE, you should polyfill this out: Paul Irish's matchMedia polyfill

Installation

First add the plugin to your project using npm or Yarn:

```
npm install vue-mq
# or
yarn add vue-mq
```

Usage

Setup your breakpoints

Setup is very straightforward. Just define your custom breakpoints when registering the plugin. Keys are breakpoints IDs and values are in pixels:

```
import Vue from 'vue'
import VueMq from 'vue-mq'

Vue.use(VueMq, {
  breakpoints: {
    sm: 450,
    md: 1250,
    lg: Infinity,
  }
})
```

You can also name your breakpoints after devices or anything that make sense to you:

```
Vue.use(VueMq, {
  breakpoints: {
    mobile: 450,
    tablet: 900,
    laptop: 1250,
    desktop: Infinity,
  }
})
```

Conditional rendering

Very often when dealing with responsive design you'll want to render elements conditionally. For instance, display a specific menu for mobile device only.

In order to do that, use the reactive `$mq` property which you can access inside each instance of component. Its value will always be the current breakpoint ID. You can easily check the value inside a `v-if` directive:

```
new Vue({
  template: `
    <mobile-menu v-if="$mq === 'mobile'">
    </mobile-menu>
  `,
})
```

`vue-mq` provides a shorthand for this syntax with a global component that acts as a conditional slot:

```
new Vue({
  template: `
    <mq-layout mq="mobile">
      <mobile-menu></mobile-menu>
    </mq-layout>
    <mq-layout mq="tablet+">
      <desktop-menu></desktop-menu>
    </mq-layout>
  `,
})
```

Notice the `+` sign after the breakpoint name. Use it to target the breakpoint and all the larger breakpoints as well.

Prop values

Another very common use-case is the computation of different values based on screen size. For example, let's say you want to display a responsive grid of items:

- on mobile you want 2 columns
- on tablets you want 3 columns
- on laptops you want 4 columns

In other words, you just have to pass down a prop with different values according to screen size to the exact same grid component. It will have the responsibility to display the right number of columns.

Very easy! `vue-mq` provides a global filter for mapping breakpoints to values, using declarative rules:

```
new Vue({
  template: `
    <grid-component :column="$mq | mq({
      phone: 2,
      tablet: 3,
      laptop: 4
    })">
    </grid-component>
  `,
})
```

Keep in mind that this plugin is enforcing a mobile-first approach. Values will default to the closest smaller breakpoints value if not defined explicitly. So here, if you omit the tablet rule: `tablet: 3`, it will display 2 columns in tablet layout.

In the same way, if you need values that are more complex, just move it in a computed prop:

```
new Vue({
  computed: {
    displayText() {
      return this.$mq === 'mobile'
      ? 'You are on mobile device'
      : 'You are on larger device'
    }
  },
  template: `
    <my-fancy-title></my-fancy-title>
  `,
})
```

Responsive Class

Sometimes, you also want to change style quickly via CSS. The trick is to use a breakpoint name as a computed class on the element you want to style. With a Single-File Component and PostCSS, for example, it's a breeze:

```
<template>
  <h1 class="my-title" :class="$mq">
    Born in the bayou
  </h1>
<template>

<style lang="postcss">
  .my-title {
    &.phone { font-size: 1em; }
    &.desktop { font-size: 3em; }
  }
<style>
```

Some ideas

`vue-mq` offers shorthands for common use-cases while its flexibility let you compose with media queries as you like.

Here are some ideas for advanced usage in combination with other libraries:

- portal-vue
- styletron-vue

Don't forget to check `vue-mq`'s documentation: vue-mq on Github and enjoy!