Vue.js - An Introduction

This basic cheat sheet for Vue.js is a progressive JavaScript framework for building user interfaces. It covers the core concepts, components, directives, props, computed properties, Vue Router, fetching data, and more. It is not an exhaustive guide but a quick reference to get started with Vue.js development.

Table of Contents

1. What is Vue.js?
2. Vue Components
3. Getting Started
4. File and Folder Structure
5. Vue Directives
6. Props in Vue Components
7. Computing Properties
8. Vue Icons
9. Vue Router
10. Fetching Data in Vue
11. `ref vs reactive`
12. Proxying API Requests
13. Toast Notifications
14. Conclusion

What is Vue.js?

- Javascript frame for building user interfaces and SPAs
- Simple, flexible, and incrementally adoptable
- Component-based architecture
- Created by Evan You in 2013
- Active community and ecosystem

Vue Components

- Building blocks of Vue applications
- Reusable, self-contained pieces of code
- Includes the logic/JS, template/HTML, and scoped styles/CSS
- HTML can also include dynamic data using directives

```
<script>
// JavaScript logic
</script>

<template>
  <!-- HTML Output -->
  <div>
    <h2>Hello from Vue.js</h2>
  </div>
</template>

<style scoped>
/* CSS Styles */
</style>
```

2 ways to build components:

1. **Options API (Vue 2.x)** - Object-based syntax
2. **Composition API (Vue 3.x)** - Function-based syntax

Getting Started

1. **CDN**: Include Vue.js via CDN in the script tag
2. **Vue CLI**: Install Vue CLI globally and create a new project. No longer recommended.
3. **Create Vue**: Uses Vite, which includes features like hot reload, built-in TypeScript support, and an ecosystem of plugins.
4. **Nuxt.js & Gridsome**: SSR & SSG frameworks built on top of Vue.js

CDN Setup

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <!-- VUE CDN -->
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <title>Vue Playground</title>
  </head>

  <body>
    <div id="app">
      <h1>{{ message }}</h1>

      <button @click="handleClick">Click Me</button>
    </div>

    <script>
      const app = Vue.createApp({
        data() {
          return {
            message: 'Hello from Vue Playground!',
          };
        },

        methods: {
          handleClick() {
            this.message = 'Button clicked!';
          },
        },
      });

      app.mount('#app');
    </script>
  </body>
</html>
```

- **Vue.createApp()**: Creates a new Vue instance
- **data()**: Returns an object with reactive properties
- **methods**: Contains custom methods
- **app.mount()**: Mounts the Vue instance to an HTML element
- **{{ }}**: Interpolates data properties in the template
- **@click**: Event listener directive

Setting up a new Vue Project

Run the following command to create a new Vue project:

```
npm create vue@latest my-vue-app
```

Navigate to the project directory:

```
cd my-vue-app
```

Install dependencies and start the development server:

```
npm install
npm run dev
```

File and Folder Structure

1. **package.json**: Contains project metadata and dependencies
2. **vite.config.js**: Vite configuration file used for custom settings and plugins
3. **index.html**: Main HTML file that loads the Vue app
4. **src**: Contains all the components and the source code of the Vue app
5. **src/main.js**: Entry point of the Vue app that mounts the app to the DOM

6. **src/App.vue**: Main component file that contains the app structure and is mounted to the DOM
7. **components**: Folder to store reusable components
8. **assets**: Folder to store static assets like images, fonts, styles, etc.
9. **public**: Folder to store static files that Vite does not process

Vue Directives

- Used to add dynamic behavior to HTML elements
- Prefixed with `v-` in the template
- Examples: `v-if`, `v-for`, `v-bind`, `v-on`, `v-model`, etc.

Directives Usage

Directives in the Template

```
<template>
  <div v-if="isUserLoggedIn">Welcome, {{ username }}</div>

  <ul>
    <li v-for="item in items" :key="item.id">{{ item.name }}</li>
  </ul>

  <input v-model="message" placeholder="Enter your message" />

  <button v-on:click="handleClick">Click Me</button>
</template>
```

- **v-if**: Conditional rendering
- **v-for**: List rendering
- **:key**: Unique key for list items
- **v-model**: Two-way data binding
- **v-on:click**: Event listener

Options API Example

```
<script>
  export default {
    data() {
      return {
        isUserLoggedIn: true,
        username: 'John Doe',
        items: [
          { id: 1, name: 'Item 1' },
          { id: 2, name: 'Item 2' },
          { id: 3, name: 'Item 3' },
        ],
        message: '',
      };
    },

    methods: {
      handleClick() {
        alert('Button clicked!');
      },
    },
  };
</script>
```

- **data()**: Returns an object with reactive properties
- **methods**: Contains custom methods
- **this**: Refers to the Vue instance

Composition API Example

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

  export default {
    setup() {
      const isUserLoggedIn = ref(true);
      const username = ref('John Doe');
      const items = ref([
        { id: 1, name: 'Item 1' },
        { id: 2, name: 'Item 2' },
        { id: 3, name: 'Item 3' },
      ]);
      const message = ref('');

      const handleClick = () => {
        alert('Button clicked!');
      };

      return {
        isUserLoggedIn,
        username,
        items,
        message,
        handleClick,
      };
    },
  };
</script>
```

- **ref()**: Creates a reactive reference to a value
- **setup()**: Function that returns reactive properties and methods
- **return**: Exposes the reactive properties and methods to the template
- Without `ref()`, the properties will not be reactive

Cleaner and Easier Syntax

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

  const isUserLoggedIn = ref(true);
  const username = ref('John Doe');
  const items = ref([
    { id: 1, name: 'Item 1' },
    { id: 2, name: 'Item 2' },
    { id: 3, name: 'Item 3' },
  ]);
  const message = ref('');

  const handleClick = () => {
    alert('Button clicked!');
  };
</script>
```

Props in Vue Components

- Used to pass data from parent to child components
- Declared in the child component and received as attributes
- Props can be passed as strings, numbers, objects, arrays, etc.
- Props are reactive and can be validated using type and default values
- Props can be one-way or two-way (v-model)

Parent Component

```
<template>
  <ChildComponent :title="pageTitle" :items="pageItems" />
</template>
```

Child Component

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

  const props = defineProps({
    title: String, // Prop type
    showButton: {
      // Prop with default value
      type: Boolean,
      default: true,
    },
  });

  console.log(props.title);
</script>

<template>
  <h1>{{ title }}</h1>
  <button v-if="showButton">Click Me</button>
</template>
```

- **defineProps()**: Function to define props and their types
- **props**: Object containing the prop values
- **String, Number, Boolean, Object, Array, Function, Symbol**: Prop types
- **v-if**: Conditional rendering based on the prop value

Computing Properties in Vue

- Reactive properties that update when dependent properties change
- Can be computed using a function or getter/setter
- Cached and only re-evaluated when necessary

Computed Properties Example

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

  const count = ref(0);

  // computed property
  const doubleCount = computed(() => count.value * 2);

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

<template>
  <div>
    <p>Count: {{ count }}</p>
    <p>Double Count: {{ doubleCount }}</p>
    <button @click="increment">Increment</button>
  </div>
</template>
```

- `doubleCount` - Computed property that doubles the value of `count`
- Evaluated only when `count` changes
- Similar to useEffect's dependency array in React

Vue Icons

We can add icons in a Vue project in various ways. Here is an example using primeicons:

1. Install PrimeIcons:

```
npm install primeicons
```

2. Import PrimeIcons in the main.js file:

```
import 'primeicons/primeicons.css';
```

3. Use the icons in the template:

```
<template>
  <i class="pi pi-check"></i>
</template>
```

Vue Router

- Official router for Vue.js
- Allows navigation between different views in a Vue application
- Supports nested routes, route parameters, and route guards
- Can be set up manually or can be used as default while initializing a Vue project

Installation

```
npm install vue-router
```

Basic Setup

1. Create a router instance:

src/router/index.js

```
import { createRouter, createWebHistory } from 'vue-router';
import HomeView from './views/HomeView.vue';

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView,
    },
    {
      path: '/jobs',
      name: 'jobs',
      component: JobsView,
    },
  ],
});

export default router;
```

2. Wrap the app with the router:

src/main.js

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App).use(router).mount('#app');
```

3. Create a view component:

src/views/HomeView.vue

```
<template>
  <main>
    <Hero />
    <HomeCards />
  </main>
</template>
```

4. Add the `RouterView` in the App.vue file:

src/App.vue

```
<script setup>
  import Navbar from '@/components/Navbar.vue';
  import { RouterView } from 'vue-router';
</script>

<template>
  <Navbar />
  <RouterView />
</template>
```

Router Links

- Use the `RouterLink` component to navigate between routes
- Can be styled using the `active-class` and `exact` props
- Can also be used with dynamic routes and route parameters

src/components/Navbar.vue

```
<script setup>
  import { RouterLink } from 'vue-router';
</script>

<template>
  <nav>
    <RouterLink to="/"> Home </RouterLink>
    <RouterLink to="/jobs"> Jobs </RouterLink>
  </nav>
</template>
```

Dynamic Routes

- Use route parameters to create dynamic routes

src/views/JobView.vue

```
<template>
  <div>
    <h1>Single Job Page</h1>
  </div>
</template>
```

src/router/index.js

```
import JobView from '@/views/JobView.vue';

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView,
    },
    {
      path: '/jobs',
      name: 'jobs',
      component: JobsView,
    },
    {
      path: '/jobs/:id',
      name: 'job',
      component: JobView,
    },
  ],
});
```

Not Found Page

- Create a `NotFound` component for handling 404 errors
- Add a wildcard route at the end of the routes array
- Redirect to the `NotFound` component if no route matches
- Use the `RouterLink` component to navigate to the home page

src/views/NotFound.vue

```
<template>
  <div>
    <h1>404 - Not Found</h1>
    <RouterLink to="/">Go to Home</RouterLink>
  </div>
</template>
```

src/router/index.js

```
import NotFound from '@/views/NotFound.vue';

const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL),
  routes: [
    {
      path: '/',
      name: 'home',
      component: HomeView,
    },
    {
      path: '/:catchAll(.*)',
      name: 'not-found',
      component: NoteFoundView,
    },
  ],
});
```

Highlighting Active Links

- Use `useRoute` to access the current route

src/components/Navbar.vue

```
<script setup>
  import { RouterLink, useRoute } from 'vue-router';

  const isActive = route => {
    const currentRoute = useRoute();
    return route === currentRoute.path;
  })
</script>

<template>
  <nav>
    <RouterLink to="/" :class="{ active: isActive('/') }"> Home </RouterLink>
    <RouterLink to="/jobs" :class="{ active: isActive('/jobs') }">
      Jobs
    </RouterLink>
  </nav>
</template>
```

Fetching Data in Vue

- Use the `fetch` API or `axios` to make HTTP requests in Vue
- Fetch data in the `setup()` function using the `onMounted` lifecycle hook
- Use `reactive` or `ref` to store the fetched data
- Handle loading, error, and success states

Fetching Data with Axios

```
npm install axios
```

src/views/JobsView.vue

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

  const jobs = ref([]);
  const loading = ref(false);
  const error = ref(null);

  onMounted(async () => {
    loading.value = true;
    try {
      const response = await axios.get('https://api.example.com/jobs');
      jobs.value = response.data;
    } catch (err) {
      error.value = err.message;
    } finally {
      loading.value = false;
    }
  });
</script>

<template>
  <div>
    <h1>Jobs</h1>
    <div v-if="loading">Loading...</div>
    <div v-else-if="error">{{ error }}</div>
    <ul v-else>
      <li v-for="job in jobs" :key="job.id">{{ job.title }}</li>
    </ul>
  </div>
</template>
```

`ref` VS `reactive`

- Both `ref` and `reactive` are used to create reactive properties in Vue
- `ref` is used for creating individual reactive values, for eg, separate fields for a form
- `reactive` is used for creating reactive objects, for eg, a form object with multiple fields
- `ref` is used for primitive values including boolean, strings, etc., while `reactive` is used for objects and arrays
- `ref` has a `.value` property to access the value or reassign it while `reactive` does not use `.value` and cannot be reassigned directly

`ref` Example

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

  const count = ref(0);

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

`reactive` Example

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

  const form = reactive({
    name: '',
    email: '',
    message: '',
  });

  const handleSubmit = () => {
    console.log(form.name, form.email, form.message);
  };

  // form.name = 'John Doe'; // Cannot reassign directly
</script>
```

Proxying API Requests in Vue

- Use proxying to avoid making multiple requests to the API using localhost
- Set up a proxy in the `vite.config.js` file

Proxy Setup

vite.config.js

```
export default defineConfig({
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:5000', // API server
        changeOrigin: true,
        rewrite: (path) => path.replace(/^\/api/, ''),
      },
    },
  },
});
```

Toast Notifications

- Use a toast library like `vue-toastification` to show notifications in a Vue app

- Install the library and set up the toast container

Installation

```
npm install vue-toastification@next
```

Setup

Import the toast container in the main.js file:

```
import Toast from 'vue-toastification';
import 'vue-toastification/dist/index.css';
```

Add the toast container to the app:

```
createApp(App).use(router).use(Toast).mount('#app');
```

Usage

Use the `useToast` hook to show notifications:

```
<script setup>
  import { useToast } from 'vue-toastification';

  const toast = useToast();

  const showToast = () => {
    toast.success('Notification message');
  };
</script>

<template>
  <button @click="showToast">Show Toast</button>
</template>
```

Conclusion

If you found the cheatsheet helpful please check out more of my work on yodkwtf.com or follow me on twitter. I also run a small youtube channel called Yodkwtf Academy.