Cheat Sheet for comprensive Vue.js

Comprehensive Vue.js Cheat Sheet

1. Introduction to Vue.js

Vue.js is a progressive JavaScript framework for building user interfaces. It is designed to be incrementally adoptable and focuses on the view layer.

Key Features:

- **Reactive Data Binding**
- **Component-Based Architecture**
- **Declarative Rendering**
- **Extensive Ecosystem**

---

2. Basic Concepts

Vue Instance

```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hello, Vue!'
  },
  methods: {
    greet() {
      alert(this.message);
    }
  }
});
```

Templates

```
<div id="app">
  <p>{{ message }}</p>
  <button @click="greet">Greet</button>
</div>
```

Directives

- **v-bind**: Binds an attribute to an expression.

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

- **v-if**: Conditionally renders an element.

```
<p v-if="show">This is visible</p>
```

- **v-for**: Renders a list of items.

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

- **v-on**: Attaches an event listener.

```
<button v-on:click="increment">Increment</button>
```

- **v-model**: Creates two-way data bindings.

```
<input v-model="message" />
```

---

3. Components

Component Structure

```
Vue.component('my-component', {
  template: `
    <div>
      <p>{{ message }}</p>
      <button @click="greet">Greet</button>
    </div>
  `,
  data() {
    return {
      message: 'Hello from component!'
    };
  },
  methods: {
```

```
    greet() {
      alert(this.message);
    }
  }
});
```

Props

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

Events

```
Vue.component('my-component', {
  template: '<button @click="$emit('increment')">Increment</button>'
});
```

Slots

```
Vue.component('my-component', {
  template: `
    <div>
      <slot name="header"></slot>
      <slot></slot>
    </div>
  `
});
```

---

4. Reactivity

Data Binding

```
new Vue({
  data: {
    message: 'Hello, Vue!'
  }
});
```

Computed Properties

```
new Vue({
  data: {
    firstName: 'John',
    lastName: 'Doe'
  },
  computed: {
    fullName() {
      return `${this.firstName} ${this.lastName}`;
    }
  }
});
```

Watchers

```
new Vue({
  data: {
    question: ''
  },
  watch: {
    question(newQuestion) {
      this.getAnswer(newQuestion);
    }
  }
});
```

---

5. Lifecycle Hooks

- **beforeCreate**: Before the instance is initialized.
- **created**: After the instance is created.
- **beforeMount**: Before the instance is mounted.
- **mounted**: After the instance is mounted.
- **beforeUpdate**: Before the instance is updated.
- **updated**: After the instance is updated.
- **beforeDestroy**: Before the instance is destroyed.
- **destroyed**: After the instance is destroyed.

---

6. Routing

Vue Router Basics

```
const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About }
];

const router = new VueRouter({
  routes
});

const app = new Vue({
  router
}).$mount('#app');
```

Navigation

```
<router-link to="/">Home</router-link>
<router-link to="/about">About</router-link>
```

Route Parameters

```
const routes = [
  { path: '/user/:id', component: User }
];
```

---

7. State Management

Vuex Basics

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state) {
      state.count++;
    }
  },
  actions: {
    incrementAsync({ commit }) {
      setTimeout(() => {
        commit('increment');
```

```
      }, 1000);
    }
  }
});
```

State, Getters, Mutations, Actions

- **State**: The source of truth.
- **Getters**: Computed properties for the store.
- **Mutations**: Methods to change the state.
- **Actions**: Methods to commit mutations.

---

8. Forms and User Input

Form Handling

```
<form @submit.prevent="submitForm">
  <input v-model="form.name" />
  <button type="submit">Submit</button>
</form>
```

Two-Way Binding

```
<input v-model="message" />
```

Validation

```
new Vue({
  data: {
    form: {
      name: ''
    }
  },
  methods: {
    submitForm() {
      if (this.form.name.trim() === '') {
        alert('Name is required');
        return;
      }
      // Submit form
    }
```

```
  }
});
```

---

9. Styling and CSS

Scoped CSS

```
<style scoped>
  p {
    color: blue;
  }
</style>
```

CSS Modules

```
<style module>
  .red {
    color: red;
  }
</style>
```

Dynamic Styling

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

---

10. Advanced Topics

Mixins

```
const myMixin = {
  created() {
    console.log('Mixin created');
  }
};

new Vue({
  mixins: [myMixin]
});
```

Custom Directives

```
Vue.directive('focus', {
  inserted(el) {
    el.focus();
  }
});
```

Render Functions

```
new Vue({
  render(h) {
    return h('div', this.message);
  }
});
```

---

11. Tips and Tricks

Performance Optimization

- **Lazy Loading Components**: Use `import()` for dynamic imports.
- **Optimize Watchers**: Use `deep` and `immediate` options carefully.
- **Virtual DOM**: Understand how Vue's virtual DOM works.

Debugging

- **Vue Devtools**: Use the browser extension for debugging.
- **Console Logging**: Use `console.log` in lifecycle hooks and methods.

Best Practices

- **Component Naming**: Use kebab-case for component names.
- **Single File Components**: Use `.vue` files for better organization.
- **Code Splitting**: Split code into smaller, reusable components.

This cheat sheet provides a comprehensive overview of Vue.js, covering essential concepts, components, reactivity, routing, state management, forms, styling, advanced topics, and best practices. Use this as a quick reference to build robust and efficient Vue.js applications.

By Ahmed Baheeg Khorshid

ver 1.0