Vue.js

- Introduction
  - What is Vue.js?
  - Why use Vue.js?
  - Getting started
    - Installation & Usage via NPM, Yarn or CDN
- **Templates**
  - Interpolation
  - Directives
  - Filters
- **Components**
  - Creating components
  - Props
  - Events
  - Slots
- **Vue Router**
  - Installing Vue Router
  - Creating routes
  - Navigation
- **Vuex**
  - Installing Vuex
  - State
  - Getters
  - Mutations
  - Actions
- **Vue CLI**
  - Installing Vue CLI
  - Creating a new project
  - Building and deploying

- **Advanced topics**
    - Mixins
    - Custom directives
    - Render functions
    - Server-side rendering

Introduction

What is Vue.js?

Vue.js is a progressive JavaScript framework for building user interfaces. It was created by Evan You and first released in 2014. Vue.js is designed to be easy to use and understand, with a focus on simplicity and flexibility.

Why use Vue.js?

Vue.js has several advantages over other JavaScript frameworks:

- **Easy to learn:** Vue.js has a gentle learning curve and can be easily integrated into existing projects.
- **Lightweight:** Vue.js is a lightweight framework, with a small file size and fast performance.
- **Flexible:** Vue.js can be used for building small to large-scale applications, and can be easily integrated with other libraries and frameworks.
- **Reactive:** Vue.js uses a reactive data binding system, which makes it easy to build dynamic and responsive user interfaces.

Getting started

Installation

- **CDN**
  To get started with Vue.js, you can include the Vue.js library in your HTML file using a script tag:

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

- **NPM**
  To install Vue.js using npm, you can run the following command in your terminal or command prompt:

```
npm install vue
```

- **Yarn**
  To install Vue.js using yarn, you can run the following command:

```
yarn add vue
```

Usage

Once you've installed Vue.js, you can create a new Vue instance:

```
import Vue from 'vue';

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

> **Note:** You don't need to import vue when adding it via CDN in your project.

This will create a new Vue instance and bind it to an element with the ID "app". The `data` property defines a message variable that can be used in the HTML template. In this example, the message variable is set to "Hello, Vue!".

To use the message variable in the HTML template, you can use the double curly brace syntax:

```
<div id="app">
  {{ message }}
</div>
```

Templates

Interpolation

In Vue.js, you can use interpolation to display data in your HTML templates. Interpolation is done using double curly braces `({{ }})`.
Here's an example:

```
<div id="app">
  {{ message }}
</div>
```

This code displays the value of the `message` variable in the HTML template.

Directives

Vue.js provides several built-in directives that you can use to manipulate the DOM, handle events, and conditionally render elements.

Here are some examples:

- `v-bind`: Binds an element's attribute to a data property. For example, `v-bind:href="url"` binds the `href` attribute to the `url` data property.
- `v-if`: Conditionally renders an element based on a condition. For example, `v-if="show"` only renders the element if the `show` data property is true.
- `v-for`: Renders a list of elements based on an array. For example, `v-for="item in items"` renders an element for each item in the `items` array.

Here's an example that uses the `v-for` directive to render a list of items:

```
<ul>
  <li v-for="item in items">{{ item }}</li>
</ul>
```

This code renders an unordered list with a list item for each item in the `items` array.

Filters

Vue.js provides filters that you can use to format data in your templates. Filters are added to an expression using the pipe (`|`) symbol.

Here's an example:

```
<div id="app">
  {{ message | capitalize }}
</div>
```

This code uses the `capitalize` filter to capitalize the `message` variable before displaying it in the HTML template.

To define a filter, you can use the `Vue.filter` method:

```
Vue.filter('capitalize', function(value) {
  if (!value) return '';
  value = value.toString();
  return value.charAt(0).toUpperCase() + value.slice(1);
});
```

This code defines a `capitalize` filter that capitalizes the first letter of a string.

Components

In Vue.js, components are reusable and self-contained blocks of code that can be used to build complex user interfaces. Components can be nested inside other components, allowing you to create a hierarchy of components that make up your application.

Creating components

To create a component in Vue.js, you can use the `Vue.component` method:

```
Vue.component('my-component', {
  template: '<div>{{ message }}</div>',
  data: function() {
    return {
      message: 'Hello, Vue!'
    }
  }
});
```

This code creates a new component called `my-component`. The `template` property defines the HTML template for the component, and the data property defines the `data` that the component uses.

To use the component in your HTML template, you can use the component's tag name:

```
<div id="app">
  <my-component></my-component>
</div>
```

This code renders the `my-component` component inside the `#app` element.

Props

Props are a way to pass data from a parent component to a child component. To define `props` for a component, you can use the props property:

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

This code defines a `message` prop for the `my-component` component. To pass data to the component, you can use the prop's name as an attribute:

```
<div id="app">
  <my-component message="Hello, Vue!"></my-component>
</div>
```

This code passes the `message` prop with the value "Hello, Vue!" to the `my-component` component.

Events

Events are a way for child components to communicate with parent components. To emit an event from a child component, you can use the `$emit` method:

```
Vue.component('my-component', {
  template: '<button @click="onClick">Click me</button>',
  methods: {
    onClick: function() {
      this.$emit('button-clicked');
    }
  }
});
```

This code defines a `my-component` component with a button that emits a `button-clicked` event when clicked.

To listen for the event in the parent component, you can use the `v-on` directive:

```
<div id="app">
  <my-component v-on:button-clicked="onButtonClicked"></my-component>
</div>
```

This code listens for the `button-clicked` event and calls the `onButtonClicked` method in the parent component.

Slots

Slots are a way to pass content from a parent component to a child component. To define a `slot` in a child component, you can use the slot element:

```
<template>
  <div>
    <h1><slot name="title"></slot></h1>
    <div><slot></slot></div>
  </div>
</template>
```

This code defines a `slot` element with a `name` attribute. The `name` attribute allows you to pass content to a specific `slot`.

To use the slot in the parent component, you can use the slot element with the `name` attribute:

```
<my-component>
  <template v-slot:title>
    My Title
  </template>
  My Content
</my-component>
```

This code passes the content "My Title" to the `title` slot and the content "My Content" to the default slot.

Vue Router

Vue Router is the official router for Vue.js. It allows you to build single-page applications with multiple views and URLs.

Installation

To install Vue Router you can run the following command in your terminal or command prompt:

```
npm install vue-router
```

or via yarn:

```
yarn add vue-router
```

Basic usage

To use Vue Router in your Vue.js application, you need to create a router instance and define your routes:

```
import Vue from 'vue';
import VueRouter from 'vue-router';

Vue.use(VueRouter);

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

const router = new VueRouter({
  routes
});

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

This code creates a new router instance with three routes: `/`, `/about`, and `/contact`. Each route is associated with a component.

To display the router's views in your HTML template, you can use the `router-view` component:

```
<div id="app">
  <router-view></router-view>
</div>
```

This code displays the router's views inside the `#app` element.

Navigation

To navigate between routes in your Vue.js application, you can use the `router-link` component:

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

This code creates three links that navigate to the `/`, `/about`, and `/contact` routes.

You can also navigate programmatically using the `$router` object:

```
this.$router.push('/about');
```

This code navigates to the `/about` route.

Route parameters

Vue Router allows you to define dynamic routes with parameters. To define a route with a parameter, you can use a colon (`:`) followed by the parameter name:

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

This code defines a route with a `id` parameter.

To access the parameter in your component, you can use the `$route` object:

```
export default {
  mounted() {
    console.log(this.$route.params.id);
  }
}
```

This code logs the ~ parameter to the console.

Nested routes

Vue Router allows you to define nested routes, where a route's component contains its own router-view:

```
const routes = [
  {
    path: '/user/:id',
    component: User,
    children: [
      { path: 'profile', component: Profile },
      { path: 'settings', component: Settings }
    ]
  }
];
```

This code defines a route with a `User` component that contains two nested routes: `/user/:id/profile` and `/user/:id/settings`.

To display the nested routes in your HTML template, you can use the nested `router-view` components:

```
<router-view></router-view>
<router-view name="profile"></router-view>
<router-view name="settings"></router-view>
```

This code displays the `User` component's view and the nested `Profile` and `Settings` components' views.

Vuex

Vuex is a state management pattern and library for Vue.js applications. It provides a centralized store for managing the state of your application, making it easier to manage and share data between components.

Installation

To install Vuex you can run the following command in your terminal or command prompt:

```
npm install vuex
```

or via yarn:

```
yarn add vuex
```

Basic usage

To use Vuex in your Vue.js application, you need to create a store instance and define your state:

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state) {
      state.count++;
    }
  }
});

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

This code creates a new store instance with a `count` state property and an `increment` mutation.

To access the store's state in your components, you can use the `$store` object:

```
export default {
  computed: {
    count() {
      return this.$store.state.count;
    }
  },
  methods: {
    increment() {
      this.$store.commit('increment');
    }
  }
}
```

This code defines a computed property that returns the store's `count` state property and a method that commits the `increment` mutation.

Mutations

Mutations are the only way to change the state in a Vuex store. To define a mutation, you can use the `mutations` property:

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state) {
      state.count++;
    },
    decrement(state) {
      state.count--;
    }
  }
});
```

This code defines two mutations: `increment` and `decrement`.

To commit a mutation in your components, you can use the `$store.commit` method:

```
this.$store.commit('increment');
```

This code commits the `increment` mutation.

Actions

Actions are used to perform asynchronous operations and commit mutations. To define an action, you can use the `actions` property:

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state) {
      state.count++;
    }
  },
  actions: {
    incrementAsync(context) {
      setTimeout(() => {
        context.commit('increment');
      }, 1000);
    }
  }
});
```

This code defines an `incrementAsync` action that commits the `increment` mutation after a delay.

To dispatch an action in your components, you can use the `$store.dispatch` method:

```
this.$store.dispatch('incrementAsync');
```

This code dispatches the `incrementAsync` action.

Getters

Getters are used to compute derived state based on the store's state. To define a getter, you can use the `getters` property:

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state) {
      state.count++;
    }
  },
  getters: {
    doubleCount(state) {
      return state.count * 2;
    }
  }
});
```

This code defines a `doubleCount` getter that returns the store's `count` state property multiplied by 2.

To access a getter in your components, you can use the `$store.getters` object:

```
export default {
  computed: {
    doubleCount() {
      return this.$store.getters.doubleCount;
    }
  }
}
```

This code defines a computed property that returns the `doubleCount` getter.

Vue CLI

Vue CLI is a command-line interface for scaffolding Vue.js projects. It provides a set of tools and configurations to help you quickly set up and develop Vue.js applications.

Installation

To install Vue CLI you can run the following command in your terminal or command prompt:

```
npm install -g @vue/cli
```

or via yarn:

```
yarn global add @vue/cli
```

Creating a new project

To create a new Vue.js project using Vue CLI, you can run the following command:

```
vue create my-project
```

This command creates a new project called `my-project` in the current directory.

Vue CLI will prompt you to select a preset for your project. You can choose from a default preset, a manually configured preset, or a remote preset.

Running the development server

To run the development server for your Vue.js project, you can run the following command:

```
npm run serve
```

This command starts the development server and opens your application in a web browser.

Building for production

To build your Vue.js project for production, you can run the following command:

```
npm run build
```

This command builds your application and generates a production-ready bundle in the `dist` directory.

Plugins

Vue CLI provides a set of plugins that you can use to add additional functionality to your project. To install a plugin, you can use the `vue add` command:

```
vue add plugin-name
```

This command installs the specified plugin and updates your project's configuration.

Configuration

Vue CLI provides a set of configuration files that you can use to customize your project's settings. These files are located in the `src` directory.

To customize your project's configuration, you can modify the files in the `src` directory or create a new configuration file.

Plugins

Vue CLI provides a set of plugins that you can use to add additional functionality to your project. To install a plugin, you can use the `vue add` command:

```
vue add plugin-name
```

This command installs the specified plugin and updates your project's configuration.

Configuration

Vue CLI provides a set of configuration files that you can use to customize your project's settings. These files are located in the `src` directory.

To customize your project's configuration, you can modify the files in the `src` directory or create a new configuration file.

Advanced Topics

Vue.js provides a wide range of advanced features and techniques that can help you build complex and powerful applications. Here are some of the most important advanced topics in Vue.js:

Render functions

Render functions are a way to create Vue.js components programmatically. They allow you to define the structure and behavior of a component using JavaScript code instead of HTML templates.

To define a render function, you can use the `createElement` method:

```
export default {
  render(createElement) {
    return createElement('div', {
      attrs: {
        id: 'app'
      }
    }, [
      createElement('h1', 'Hello, Vue!'),
      createElement('p', 'This is a render function.')
    ]);
  }
}
```

This code defines a component with a render function that creates a `div` element with an `id` attribute and two child elements: an `h1` element and a `p` element.

Mixins

Mixins are a way to share code between Vue.js components. They allow you to define reusable behavior that can be applied to multiple components.

To define a mixin, you can use the `mixins` property:

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

export default {
  mixins: [myMixin],
  created() {
    console.log('Component created.');
  }
}
```

This code defines a mixin with a `created` hook and a component that uses the mixin. When the component is created, both the mixin's `created` hook and the component's `created` hook are called.

Custom directives

Directives are a way to add custom behavior to HTML elements in Vue.js. They allow you to define custom attributes that can be used to modify the behavior of an element.

To define a custom directive, you can use the `directive` method:

```
Vue.directive('my-directive', {
  bind(el, binding) {
    el.style.color = binding.value;
  }
});
```

This code defines a custom directive called `my-directive` that sets the color of an element based on its binding value.

To use the directive in your HTML template, you can use the `v-my-directive` syntax:

```
<div v-my-directive="'red'">Hello, Vue!</div>
```

This code applies the `my-directive` directive to a `div` element with a binding value of `'red'`.

Filters

Filters are a way to format data in Vue.js. They allow you to define custom functions that can be used to modify the output of a data property.

To define a filter, you can use the `filter` method:

```
Vue.filter('uppercase', function(value) {
  return value.toUpperCase();
});
```

This code defines a filter called `uppercase` that converts a string to uppercase.

To use the filter in your HTML template, you can use the `{{ }}` syntax:

```
<div>{{ message | uppercase }}</div>
```

This code applies the `uppercase` filter to a `message` data property.

Transitions

Transitions are a way to add animations to Vue.js components. They allow you to define custom animations that can be applied to a component when it is inserted, updated, or removed from the DOM.

To define a transition, you can use the `transition` component:

```html
<transition name="fade">
  <div v-if="show">Hello, Vue!</div>
</transition>
```

This code defines a transition called `fade` that fades in and out a `div` element when it is inserted or removed from the DOM.