ARYATECHNO

Vue.js Exercises

Topics : Vue
Written on January 11, 2024

Exercise 1: Simple Binding

1. Create a Vue instance with a data property called `message` and bind it to an `<h1>` element in the template.

Exercise 2: Two-Way Binding

2. Create a Vue instance with a data property called `inputText` and bind it to an `<input>` element. Display the value of `inputText` below the input field.

Exercise 3: Conditional Rendering

3. Create a Vue instance with a data property called `showMessage` and use it to conditionally render a `<p>` element with the text "Hello, Vue!".

Exercise 4: List Rendering

4. Create a Vue instance with a data property called `fruits` containing an array of fruits. Use `v-for` to render an unordered list (`<ul>`) with list items (`<li>`) for each fruit.

Exercise 5: Event Handling

5. Create a Vue instance with a method called `handleButtonClick`. Use `v-on` to bind this method to a button's click event. When the button is clicked, log a message to the console.

Exercise 6: Computed Properties

6. Create a Vue instance with data properties `num1` and `num2`. Add a computed property called `sum` that calculates the sum of `num1` and `num2` and displays it in the template.

Exercise 7: Component Communication

7. Create a parent component that includes a child component. Pass a prop from the parent to the child with a custom message, and have the child component display this message.

Exercise 8: Form Handling

8. Create a Vue instance with a form containing input fields for name and email. Use `v-model` for two-way binding and add a method to handle the form submission, logging the form data to the console.

Exercise 9: Routing

9. Set up a simple Vue Router with two routes: one for a home page and another for an about page. Create components for each page and navigate between them using `<router-link>`.

Exercise 10: Vuex

10. Set up a basic Vuex store with a state property called `counter`. Create mutations to increment and decrement the counter. Display the counter value in a component and allow users to interactively change it.

Exercise 11: Dynamic Styling

11. Create a Vue instance with a data property called `isHighlighted`. Use this property to dynamically apply a CSS class that highlights a `<div>` element when the value is true.

Exercise 12: Lifecycle Hooks

12. Create a Vue instance with lifecycle hooks (`created`, `mounted`, `updated`, `destroyed`). Log a message to the console in each hook to observe the order of execution.

Exercise 13: Watchers

13. Set up a Vue instance with a data property called `watchedValue`. Create a watcher that logs a message to the console whenever `watchedValue` changes.

Exercise 14: Custom Directives

14. Create a custom directive called `v-uppercase` that transforms the text content of an element to uppercase when used in the template.

Exercise 15: Slots

15. Create a parent component with a slot and a child component that uses this slot. Pass content from the parent component to the child using the slot.

Exercise 16: Form Validation

16. Implement a simple form with input fields for username and password. Add form validation to ensure that both fields are filled before submission.

Exercise 17: Axios Integration

17. Set up a Vue instance that makes an asynchronous HTTP request using Axios. Fetch data from a mock API and display it in the template.

Exercise 18: Vuex Actions

18. Expand the Vuex store from Exercise 10. Create an action to asynchronously increment the counter and commit the mutation.

Exercise 19: Dynamic Route Parameters

19. Set up dynamic route parameters in your Vue Router. Create a route that takes an ID parameter and displays details for an item with that ID.

Exercise 20: Transitions and Animations

20. Implement a transition or animation effect in your Vue.js application. For example, fade in/fade out or slide in/slide out when elements are added or removed.

Exercise 21: Dynamic Components

21. Create a Vue instance with a data property called `currentComponent` and use it to dynamically render different components based on user input.

Exercise 22: Global Event Bus

22. Implement a global event bus using a new Vue instance. Use it to communicate between two unrelated components.

Exercise 23: Scoped Slots

23. Create a parent component that passes data to a child component through a scoped slot. The child component should display the data in a custom-styled manner.

Exercise 24: Unit Testing

24. Write unit tests for a simple Vue component using a testing library such as Jest or Mocha. Test different aspects, including methods, computed properties, and DOM interactions.

Exercise 25: Vue CLI Project

25. Create a new Vue.js project using Vue CLI. Configure the project structure, add components, and explore the development server.

Exercise 26: Nuxt.js Project

26. Set up a basic Nuxt.js project with two pages. Use layouts, asyncData, and fetch to enhance the pages.

Exercise 27: Vuex Modules

27. Expand the Vuex store with modules. Create separate modules for different aspects of your application's state.

Exercise 28: Scoped Styles

28. Implement scoped styles in a Vue component using the `scoped` attribute or a CSS preprocessor like Sass.

Exercise 29: Error Handling

29. Add error handling to your Vue.js application. Use a global error handler or incorporate local error handling in specific components.

Exercise 30: Vue.js and TypeScript

30. Convert an existing Vue.js project to use TypeScript. Update components, add type definitions, and explore TypeScript features in a Vue context.