Discussion about VueJs Tutorial

Week 3

Task 1 introduces Vue instance and the way to use expression. Vue instance is the root of our application which provides many optional properties to preserve data and implement methods (Jahr, n.d.). In order to perform the relationship between instance and part of DOM, the element of instance is applied by using ‘el’ property (Jahr, n.d.). The instance’s data that we defined inside ‘data’ property can be accessed by putting expression {{..}} to HTML tag. When Vue tracks the expression, it references the instance’s data and replaces that expression with the value of that data. We explore the way to bind the data to the attribute of HTML elements in task 2. Jahr (n.d.) emphasizes the term “Data Binding” that the data of the instance is the host linked to target data. The key point is to make HTML attributes dynamic. Depending on the condition, Jahr (n.d.) identifies in task 3 some conditional syntaxes such as v-if, v-else… to render the variant that we want to display. V-show is a useful syntax to toggle the visible of variant. In task 4, we need to use v-for to visualize each product detail stored in array. This syntax iterates each detail and renders it to the webpage.

Week 4

Jahr (n.d.) introduces event handling in task 5 to listen for DOM events. The button is listening on click events with v-on:click syntax, and the addToCart method is triggered for each button clicked. Similarly, we use mouseover event to handle the mouse hovering over the variant. The event can be tracked conveniently by using the shorthand “@”. Another way to style our variants is using dynamic style binding in task 6. We learned to use variant’s color to create the style of div component. The goal is to hover over the color box to witness the image matching with that color, instead of hovering over the text. We also prevent users from clicking button for adding to cart by checking inStock flag. If inStock is false, we would bind “disabled” attribute to the button and gray it out. A new definition which we explore in task 7 is computed properties. Computed properties help us to calculate the value rather than store it. In first case, we prefer to display “Vue Mastery Socks” by using computed properties to create a computed property called “title”, then concatenating “brand” and “product” string once “title” is assigned. In complex case, the code refactor replaces image with selectedVariant in data, allowing dynamic updates based on hovered variant indices. By using computed properties for image and inStock, the application efficiently manages dependencies, enabling the display of variant-specific images and stock status while maintaining the button's conditional behavior.

Week 5

Components introduced in task 8 are powerful and reuseable blocks of code, helping us manage and maintain the code block effectively. We initialize the component by registering the name (“product”) and using “template” property to build our product structure. The template has exactly one root element containing HTML tags to be rendered. The component also provides “data”, “methods” and “computed” properties. Finally, we call “product” component to visualize the content in webpage. To pass data from a parent to a child component in Vue, use **props**. Define the props object in the child component to specify the expected data type and attributes, then pass the parent data to the child via a custom attribute bound with “:”. In task 9, to communicate from a child component to its parent in Vue, we use event emitters. The child component emits an event with $emit, optionally sending data, while the parent listens for the event using v-on or the shorthand @, enabling event-driven communication. In our work, after applying $emit, the addToCart method in the child component emits the add-to-cart event along with the selected product's variantId. This signals the parent component that a product has been added to the cart. We implement the form in task 10 to collect user input and handle form validation. The child component product-review collects user input through form elements bound with v-model for two-way data binding. When the form is submitted, the onSubmit method creates a productReview object with the input data and emits a review-submitted event with this object. The parent component listens for this event and invokes its addReview method to add the received review object to its reviews array. We could use “required” attributes or customize our validation.

Refs

Jahr, A. (n.d.) *Intro to VueJs*. https://www.vuemastery.com/courses/intro-to-vue-js.