Advanced Web Programming

CAT 3: The state is key

Questions

Exercise 1. (0.5 points)

Vue implements a notation transformation mechanism for event names. What kind of notations is recommended to use for emitting events and listening for them in the parent component? Give an illustrative example.

Vue 3 introduced a new event name notation transformation mechanism called "kebab-case" and "camelCase" auto conversion. This means that when emitting events in child components, you can use either kebab-case or camelCase for event names, and Vue will automatically convert them as needed. Also, there're two ways to capture events in Vue 3: v-on and watch

- v-on: Used to listen to DOM events such as click, mouseover, etc…

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

- $watch(): Performs actions when the state of a component changes. Can be used in conjunction with v-on

```
watch: {
  // cada vez que la pregunta cambie, esta función será ejecutada
  question: function (newQuestion, oldQuestion) {
    this.answer = 'Esperando que deje de escribir...'
    this.debouncedGetAnswer()
  }
},
```

Exercise 2. (0.5 points)

What does the `emits` element include in the scope of a component? What two structures can be used to detail its members? Give an example.

In Vue 3, the emits option is used in the context of the parent-child communication to explicitly declare the events that a child component can emit to its parent. It helps to document and enforce a contract regarding the events that the child component may trigger.

The emits option can include an array or an object to detail its members. These structures are used to specify the events that a component can emit.

Using an array:

```
// ChildComponent.vue

export default {
  emits: ['child-button-clicked'],
  // ...
  methods: {
    emitEvent() {
      this.$emit('child-button-clicked', 'Hello from ChildComponent');
    }
  }
}
```

Using an object:

```
// ChildComponent.vue

export default {
  emits: {
    'child-button-clicked': (payload) => {
      return typeof payload === 'string';
    }
  },
  // ...
  methods: {
    emitEvent() {
      this.$emit('child-button-clicked', 'Hello from ChildComponent');
    }
  }
}
```

Exercise 3. (0.5 points)

Given the following component:

```
<script>
export default {
  name: "App",
    data: () => ({
    items: [
       { body: 'Scoped Slots Guide', username: 'Evan You', likes: 20 },
       { body: 'Vue Tutorial', username: 'Natalia Tepluhina', likes: 10 }
    ]
  })
};
</script>

<template>
  <ul>
   <li></li>
  </ul>
</template>
```

Populate the component template using a directive that generates an element `that` for each item in the list items. The `li` element should have the following format:

`Scoped Slots Guide` `::` <u>`Evan You`</u> `::` *`20`*

What directive can we use to query if the array `items` has elements, i.e., it is not empty?

```
<template>
<ul>
<li v-for="item in items" :key="item.id">
<slot :item="item">
<div>
<strong>{{ item.body }}</strong>
<span> :: </span>
<u>{{ item.username }}</u>
<span> :: </span>
<i>{{ item.likes }}</i>
</div>
</slot>
</li>
</ul>
```

```
</template>
```

Exercise 4. (0.5 points)

Given the following component:

```
<template>
  <div>
    <p>Is there any pending task?</p>

  </div>
</template>

<script>
export default {
  name: "App",
  data: () => ({
      module: {
        name: 'Messages management',
        pendingTasks: [
          { id:1, desc: 'Review post 3', priority: 'high' },
          { id:2, desc: 'Sort posts', priority: 'medium' },
          { id:3, desc: 'Check for new updates', priority: 'low' }
        ]
      }
    })
  };
</script>
```

Develop a function `hasPendingTasks` that returns `true` if the array `pendingTasks` has elements and `false` otherwise. Complete the template of the component so that it reports whether or not there are pending tasks.

Note: It must be a reactive function, so if the content of the array is modified `pendingTasks,` the information shown in the template should be automatically updated.

```
<template>
<div>
<p>Is there any pending task?</p>
<p v-if="hasPendingTasks">Yes</p>
<p v-else>No</p>
</div>
</template>

<script>
export default {
name: "App",
data: () => ({
module: {
name: "Messages management",
pendingTasks: [
{ id: 1, desc: "Review post 3", priority: "high" },
{ id: 2, desc: "Sort posts", priority: "medium" },
{ id: 3, desc: "Check for new updates", priority: "low" },
],
},
}),
computed: {
hasPendingTasks() {
return this.module.pendingTasks.length > 0;
},
},
};
</script>
```

Exercise 5. (0.5 points)

A. What is a computed property? Give two examples.
    a. In Vue 3, a computed property is a special type of property that is derived from one or more reactive properties. Computed properties are declared in the computed option of a component, and their values are calculated based on the values of other reactive properties. Computed properties are reactive themselves, meaning they will automatically update when the dependencies they rely on change.

B. What differentiates a computed property from a "watched" property? Illustrate it with an example.
    a. In Vue 3, both computed properties and watched properties are used to reactively respond to changes in the underlying data, but they serve different purposes and have different use cases. Watchers are used when you need to perform some asynchronous or side-effectful operation in response to changes in data. They are

suitable for more complex scenarios where you need to perform custom logic in response to changes.