Módulo 1: Introdução ao Vue.js

- Vuejs computed e watch e suas peculiaridades

Em Vue.js, tanto as **propriedades computadas** (`computed`) quanto os **observadores** (`watch`) são usados para acompanhar mudanças nos dados e realizar ações com base nessas mudanças. No entanto, eles têm finalidades diferentes, e cada um tem suas peculiaridades. Vamos entender cada um detalhadamente.

1. Propriedades Computadas (`computed`)

As propriedades **computadas** são como campos derivados. Elas são baseadas em outras propriedades do componente, e são recalculadas **somente quando as dependências mudam**. Ou seja, se a propriedade dependente não mudar, o Vue não recalcula o valor da propriedade computada, resultando em uma otimização de performance.

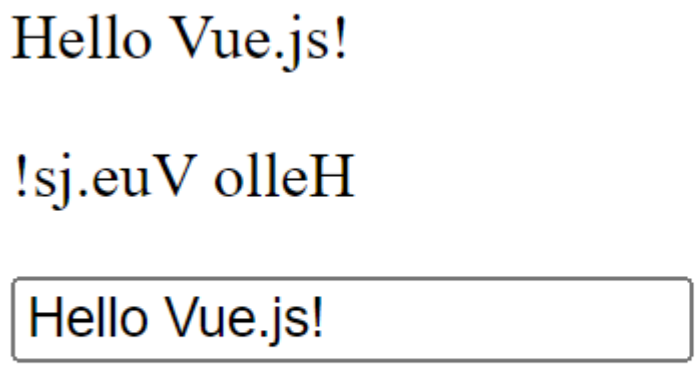
Exemplo Prático:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vue.js Example</title>
</head>
<body>

    <div id="app">
        <p>{{ message }}</p>
        <p>{{ reversedMessage }}</p>
        <input v-model="message" />
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
    <script>
      var app = new Vue({
        el: '#app',
        data: {
          message: 'Hello Vue.js!'
        },
        computed: {
          // Propriedade computada que inverte a string da mensagem
          reversedMessage() {
            return this.message.split('').reverse().join('');
          }
        }
      });
    </script>

</body>
</html>
```



Explicação:

- A `reversedMessage` é uma **propriedade computada**. Ela depende de `message` e será recalculada apenas quando o valor de `message` mudar.
- O Vue.js faz o cache do valor da propriedade computada, ou seja, enquanto `message` não for alterado, `reversedMessage` não será recalculada, otimizando o desempenho.

Peculiaridade:

- As propriedades computadas são altamente eficientes porque só são recalculadas quando as dependências (neste caso, `message`) mudam.
- Usam cache, então mesmo que a função seja chamada várias vezes no template, o valor só é recalculado quando necessário.

2. Observadores (`watch`)

Os **observadores** (`watch`) são usados para executar lógica ou efeitos colaterais em resposta a mudanças específicas de variáveis reativas. Eles são ideais para realizar tarefas mais complexas ou que precisam acontecer de forma assíncrona, como chamadas a uma API ou animações, quando um valor específico muda.

Exemplo Prático:

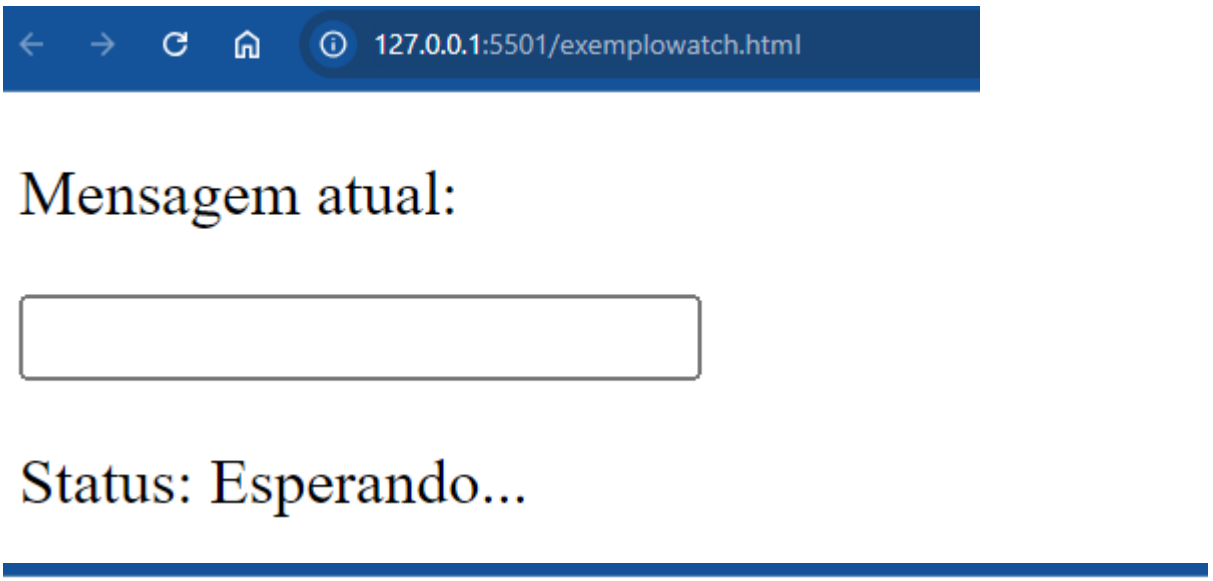
exemplowatch.html > html > body

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>exemplo com watch</title>
</head>
<body>

    <div id="app">
      <p>Mensagem atual: {{ message }}</p>
      <input v-model="message" />
      <p>Status: {{ status }}</p>
    </div>

  <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
  <script>
  var app = new Vue({
    el: '#app',
        data: {
        message: '',
            status: 'Esperando...'
        },
    watch: {
      // Observador que "escuta" mudanças na propriedade `message`
      message(newValue, oldValue) {
        this.status = `Mensagem mudou de '${oldValue}' para '${newValue}'`;
      }
    }
  });
  </script>

</body>
</html>
```



Explicação:

- No exemplo acima, o **observador** está "assistindo" a propriedade `message`. Quando `message` muda, a função associada ao observador é executada.
- A função recebe dois argumentos: o novo valor (`newValue`) e o valor anterior (`oldValue`).
- Usamos o `watch` para atualizar o status toda vez que `message` é alterada, mas isso poderia ser qualquer lógica personalizada, como uma chamada de API, validação, etc.

Peculiaridade:

- O **observador** é útil quando você precisa reagir a mudanças em tempo real, especialmente quando essas mudanças resultam em efeitos colaterais, como chamadas a APIs ou animações.

- **Não usa cache**, diferente das propriedades computadas. Isso significa que sempre que o valor observado muda, o código do observador é executado.

Diferenças Importantes:

- **Uso de cache**: Propriedades computadas são otimizadas com cache. Elas são recalculadas somente quando suas dependências mudam. Observadores, por outro lado, executam a lógica toda vez que há uma mudança, sem cache.
- **Simplicidade**: Propriedades computadas são mais simples quando você precisa derivar valores com base em outras propriedades. Observadores são mais adequados para lógica complexa e assíncrona.
- **Finalidade**: Use `computed` para **derivar** dados e `watch` para **reagir** a mudanças e realizar tarefas específicas.

Quando usar cada um:

- **`computed`**: Use quando você precisa derivar ou transformar dados que dependem de outros valores reativos. Isso é útil quando o valor resultante será usado diretamente no template, sem precisar de lógica extra.
- **`watch`**: Use quando você precisa executar **lógica complexa ou assíncrona** em resposta a uma mudança de valor (como chamadas de API, validações avançadas ou salvar dados no backend).

Exemplo Completo Combinando **Computed** e **Watch**:

exemplowatch&COMPUTED.html

exemplowatch&COMPUTED.html > html > body > script > app > watch

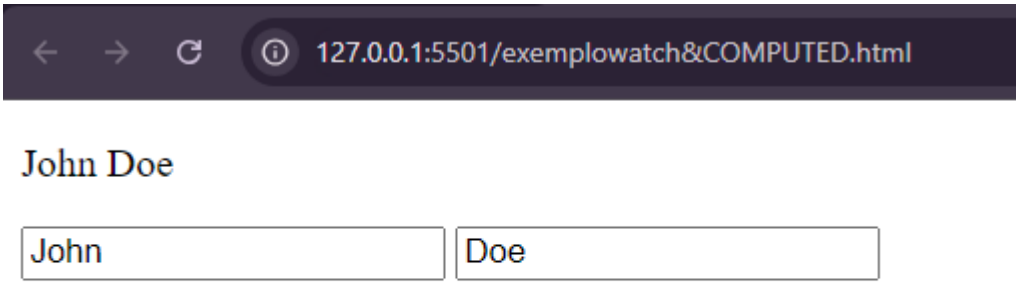
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>exemplo com watch</title>
</head>
<body>

  <div id="app">
    <p>{{ fullName }}</p>
    <input v-model="firstName" placeholder="First Name" />
    <input v-model="lastName" placeholder="Last Name" />
    <p>{{ apiResponse }}</p>
  </div>

```

```
    <script src="https://cdn.jsdelivr.net/npm/vue@2.7.14/dist/vue.js"></script>
    <script>
    var app = new Vue({
      el: '#app',
          data: {
            firstName: 'John',
            lastName: 'Doe',
            apiResponse: ''
          },
          computed: {
        // Propriedade computada que retorna o nome completo
        fullName() {
          return `${this.firstName} ${this.lastName}`;
        }
      },
      watch: {
      // Observa mudanças no `fullName` e faz uma chamada de API fictícia
      fullName(newName) {
        this.fakeApiCall(newName);
      }
    },
    methods: {
      fakeApiCall(name) {
        // Simulando uma chamada de API e alterando apiResponse
        this.apiResponse = `API chamada com o nome: ${name}`;
       }
     }
    });
    </script>

</body>
</html>
```



Neste exemplo, o `fullName` é uma propriedade computada que combina `firstName` e `lastName`, enquanto o `watch` executa uma função sempre que `fullName` muda, simulando uma chamada a uma API.