Parte 1: Questões Teóricas (15 questões)

1. Explique o que é o jsFiddle e qual a sua utilidade no desenvolvimento com Vue.js.

O jsFiddle é uma ferramenta online que permite testar, compartilhar e depurar códigos em HTML, CSS e JavaScript diretamente no navegador. No contexto do Vue.js, ele é amplamente utilizado para criar protótipos rápidos, demonstrar funcionalidades específicas do framework e validar conceitos de forma prática e colaborativa sem a necessidade de configuração local de ambiente de desenvolvimento.

2. Descreva o processo para configurar o Vue.js no jsFiddle. Quais os passos necessários para incluir a biblioteca Vue.js?

Para configurar o Vue.js no jsFiddle, é necessário acessar o menu de configurações do editor, localizar a seção de bibliotecas externas e adicionar o link da CDN oficial do Vue.js. Após incluir a biblioteca, basta escrever o código HTML, JavaScript e CSS nos respectivos painéis do jsFiddle, permitindo que o Vue seja utilizado de forma imediata para criar exemplos funcionais.

3. O que é o "Hello World" no Vue.js e qual a sua importância no aprendizado do framework?

O "Hello World" no Vue.js é o exemplo básico que demonstra como exibir dados dinamicamente no HTML utilizando a sintaxe do framework. Sua importância reside no fato de introduzir os conceitos iniciais de reatividade e interpolação de dados, funcionando como o primeiro contato prático com o funcionamento essencial do Vue.

4. Defina o conceito de two-way data binding no Vue.js. Por que ele é uma característica importante?

O two-way data binding no Vue.js é o recurso que sincroniza automaticamente os dados entre o modelo JavaScript e a interface do usuário. Essa característica é fundamental porque reduz a necessidade de manipulação manual do DOM, tornando o desenvolvimento de formulários e componentes interativos mais intuitivo e eficiente.

5. Qual a principal função do v-for no Vue.js? Dê um exemplo de quando ele é utilizado.

A diretiva v-for é utilizada para renderizar listas de elementos dinamicamente com base em dados iteráveis, como arrays ou objetos. Por exemplo, ela é aplicada quando se deseja exibir uma lista de usuários vindos de uma API, criando automaticamente um elemento HTML para cada item retornado.

6. Explique o propósito da diretiva v-bind:key ao trabalhar com listas dinâmicas no Vue.js.

A diretiva v-bind:key atribui uma chave única para cada item renderizado em uma lista, permitindo ao Vue identificar eficientemente quais elementos foram alterados, adicionados ou removidos. Isso otimiza a atualização do DOM virtual, evitando renderizações desnecessárias e melhorando a performance da aplicação.

7. O que é a reatividade no Vue.js e como ela simplifica o desenvolvimento de interfaces dinâmicas?

A reatividade no Vue.js é o mecanismo que atualiza automaticamente a interface sempre que os dados do modelo sofrem alterações. Esse recurso simplifica o desenvolvimento ao eliminar a necessidade de atualizações manuais do DOM, garantindo que a interface esteja sempre sincronizada com o estado atual da aplicação.

8. Quais são as principais vantagens de usar o ciclo de vida dos componentes no Vue.js? Cite pelo menos dois hooks e suas finalidades.

O ciclo de vida dos componentes permite maior controle sobre a criação, atualização e destruição de elementos no Vue.js. O hook mounted, por exemplo, é usado para executar código após a montagem do componente, enquanto o hook beforeDestroy permite realizar limpezas antes que o componente seja removido do DOM.

9. Diferencie os métodos JavaScript push e splice ao manipular arrays no Vue.js. Como o Vue detecta alterações feitas com esses métodos?

O método push adiciona novos elementos ao final de um array, enquanto splice pode inserir ou remover elementos em posições específicas. O Vue detecta alterações feitas com ambos os métodos porque eles acionam internamente seus mecanismos reativos, garantindo que o DOM seja atualizado de acordo com as mudanças nos dados.

10. O que é o método Vue.set e em quais cenários ele é necessário para garantir a reatividade?

O método Vue.set é utilizado para adicionar propriedades reativas a objetos ou atualizar índices de arrays que não foram previamente declarados. Ele é necessário quando se deseja garantir que novas propriedades ou mudanças específicas sejam reconhecidas pelo sistema reativo e reflitam corretamente na interface.

11. Explique o uso dos modificadores de evento .prevent e .stop no Vue.js. Como eles afetam o comportamento de um evento do DOM?

O modificador .prevent é utilizado para chamar preventDefault e impedir o comportamento padrão de um evento, como o envio de um formulário. Já o modificador .stop chama stopPropagation, bloqueando a propagação do evento para elementos ancestrais no DOM. Ambos oferecem maior controle sobre a gestão de eventos na aplicação

12. Descreva como os modificadores de teclas, como .enter e .esc, são usados no Vue.js. Cite um exemplo prático de aplicação.

Os modificadores de teclas no Vue.js permitem associar eventos a teclas específicas do teclado. Por exemplo, um input de busca pode utilizar @keyup.enter para executar a pesquisa quando a tecla Enter for pressionada, tornando a interação mais intuitiva para o usuário.

13. Quais são as diferenças entre checkboxes, radios e selects em formulários dinâmicos no Vue.js?

Checkboxes permitem múltiplas seleções e são vinculados a arrays, radios permitem apenas uma escolha exclusiva, e selects podem ser configurados para seleção única ou múltipla. No Vue.js, todos esses elementos podem ser facilmente controlados com v-model, que mantém o estado do formulário sincronizado com os dados.

14. Por que é importante usar o atributo v-model ao criar formulários no Vue.js?

O v-model é importante porque simplifica a sincronização bidirecional entre os campos do formulário e os dados do componente. Ele elimina a necessidade de eventos adicionais para capturar e atualizar valores manualmente, tornando o desenvolvimento de formulários mais limpo e direto.

15. Explique como o Vue.js atualiza o DOM de maneira eficiente ao usar o Virtual DOM. Por que isso é vantajoso?

O Vue.js utiliza o Virtual DOM para criar uma representação leve do DOM real e comparar diferenças antes de aplicar atualizações. Essa abordagem reduz operações diretas no DOM, melhorando o desempenho e garantindo uma renderização mais rápida e eficiente das interfaces.

Parte 2: Questões Práticas (10 questões)

1. Configure um ambiente no jsFiddle para utilizar o Vue.js e crie um exemplo básico que exiba uma mensagem "Olá, Vue.js!". Inclua um botão para alterar a mensagem dinamicamente.

```
<div id="app">
  <p>{{ mensagem }}</p>
  <button @click="alterarMensagem">Alterar Mensagem</button>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
  const { createApp } = Vue;
  createApp({
    data() {
      return { mensagem: 'Olá, Vue.js!' };
    },
    methods: {
      alterarMensagem() {
        this.mensagem = 'Mensagem alterada!';
      }
    }
  }).mount('#app');
</script>
```

2. Crie um contador interativo usando o Vue.js, onde dois botões incrementem e decremente o valor do contador.

```
<div id="app">
  <p>Contador: {{ contador }}</p>
  <button @click="contador++">+</button>
  <button @click="contador--">-</button>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() {
    return { contador: 0 }
  }
}).mount('#app');
</script>
```

3. Implemente uma lista de tarefas usando v-for para renderizar os itens. Permita que o usuário adicione e remova tarefas da lista.

```
<div id="app">
  <input v-model="novaTarefa" placeholder="Nova tarefa" />
  <button @click="adicionarTarefa">Adicionar</button>
  <ul>
    <li v-for="(tarefa, index) in tarefas" :key="index">
      {{ tarefa }} <button @click="removerTarefa(index)">Remover</button>
    </li>
  </ul>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() {
    return { novaTarefa: '', tarefas: [] }
  },
  methods: {
    adicionarTarefa() {
      if (this.novaTarefa) {
        this.tarefas.push(this.novaTarefa);
        this.novaTarefa = '';
      }
    },
    removerTarefa(i) {
      this.tarefas.splice(i, 1);
    }
  }
}).mount('#app');
</script>
```

4. Crie um exemplo em que você utiliza Vue.set para atualizar dinamicamente o valor de um índice específico de um array, garantindo a reatividade.

```
<div id="app">
  <p>{{ numeros }}</p>
  <button @click="atualizarIndice">Atualizar índice 1</button>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() {
    return { numeros: [10, 20, 30] }
  },
  methods: {
    atualizarIndice() {
      this.numeros[1] = 99;
    }
  }
}).mount('#app');
</script>
```

5. Desenvolva um formulário dinâmico com Vue.js que inclua os seguintes campos:

○ Nome (input de texto).

○ Gênero (radio buttons).

○ Interesses (checkboxes).

○ Cidade (select). Exiba os dados preenchidos abaixo do formulário em tempo real.

```
<div id="app">
  <form>
    <label>Nome: <input v-model="form.nome" /></label><br/>
    <label>Gênero:
      <input type="radio" value="Masculino" v-model="form.genero" /> Masculino
      <input type="radio" value="Feminino" v-model="form.genero" /> Feminino
    </label><br/>
    <label>Interesses:
      <input type="checkbox" value="Esportes" v-model="form.interesses" /> Esportes
      <input type="checkbox" value="Música" v-model="form.interesses" /> Música
    </label><br/>
    <label>Cidade:
      <select v-model="form.cidade">
        <option>São Paulo</option>
        <option>Rio de Janeiro</option>
      </select>
    </label>
  </form>
  <pre>{{ form }}</pre>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() {
    return { form: { nome: '', genero: '', interesses: [], cidade: '' } }
  }
}).mount('#app');
</script>
```

6. Usando modificadores de evento, crie um formulário onde o comportamento padrão do botão de envio seja prevenido e, em vez disso, exiba os dados do formulário em um alerta.

```
<div id="app">
  <form @submit.prevent="enviarFormulario">
    <input v-model="nome" placeholder="Digite seu nome" />
    <button type="submit">Enviar</button>
  </form>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() { return { nome: '' }; },
  methods: {
    enviarFormulario() {
      alert(`Nome enviado: ${this.nome}`);
    }
  }
}).mount('#app');
</script>
```

7. Crie um exemplo que capture eventos de teclado no Vue.js. Quando o usuário pressionar Enter, o texto digitado deve ser salvo. Caso pressione Esc, o texto deve ser apagado.

```
<div id="app">
  <input
    v-model="texto"
    @keyup.enter="salvar"
    @keyup.esc="limpar"
    placeholder="Digite algo"
  />
  <p>Texto: {{ texto }}</p>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() { return { texto: '' }; },
  methods: {
    salvar() { alert('Texto salvo: ' + this.texto); },
    limpar() { this.texto = ''; }
  }
}).mount('#app');
</script>
```

8. Construa uma aplicação Vue.js que renderize uma lista de produtos com os seguintes dados: Nome, Preço e Quantidade. Permita que o usuário edite o preço de um produto específico dinamicamente.

```
<div id="app">
  <ul>
    <li v-for="(p, index) in produtos" :key="index">
      {{ p.nome }} -
      <input type="number" v-model="p.preco" /> (Qtd: {{ p.quantidade }})
    </li>
  </ul>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() {
    return { produtos: [
      { nome: 'Teclado', preco: 100, quantidade: 5 },
      { nome: 'Mouse', preco: 50, quantidade: 10 }
    ]}
  }
}).mount('#app');
</script>
```

9. Desenvolva uma página que utilize o ciclo de vida dos componentes para:

○ Mostrar uma mensagem de log no console ao inicializar o componente.

○ Realizar uma requisição (fictícia) simulada quando o componente for montado.

```
<div id="app">
  <p>Verifique o console!</p>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  mounted() {
    console.log("Componente montado!");
    setTimeout(() => {
      console.log("Requisição fictícia simulada concluída!");
    }, 2000);
  }
}).mount('#app');
</script>
```

10. Implemente uma interface que permita ao usuário adicionar itens a um carrinho de compras. Cada item deve ser exibido em uma tabela com a opção de removê-lo. Use v-for e v-bind:key para lidar com os dados dinamicamente.

```
<div id="app">
  <input v-model="novoItem" placeholder="Novo item" />
  <button @click="adicionarItem">Adicionar</button>
  <table border="1">
    <tr>
      <th>Item</th><th>Ação</th>
    </tr>
    <tr v-for="(item, index) in carrinho" :key="index">
      <td>{{ item }}</td>
      <td><button @click="removerItem(index)">Remover</button></td>
    </tr>
  </table>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
createApp({
  data() { return { novoItem: '', carrinho: [] }; },
  methods: {
    adicionarItem() {
      if (this.novoItem) {
        this.carrinho.push(this.novoItem);
        this.novoItem = '';
      }
    },
    removerItem(i) { this.carrinho.splice(i, 1); }
  }
}).mount('#app');
</script>
```