Questões Teóricas

1.O que é o Vuetify e quais são suas principais vantagens ao desenvolver interfaces no Vue.js?

O Vuetify é um framework de componentes baseado em Material Design para Vue.js. Ele facilita o desenvolvimento de interfaces modernas e responsivas, fornecendo uma vasta coleção de componentes prontos, como botões, cards, formulários e tabelas. Suas principais vantagens incluem a consistência visual, a economia de tempo no desenvolvimento e a alta personalização.

2. Explique o processo de configuração inicial do Vuetify em um projeto Vue.js. Quais são os passos necessários para adicioná-lo ao projeto?

O processo começa com a criação de um projeto Vue.js utilizando Vue CLI. Após isso, o Vuetify pode ser instalado via npm ou yarn, e configurado automaticamente com o comando vue add vuetify. Esse comando realiza as alterações necessárias no projeto, como a inclusão do Vuetify no arquivo main.js e a configuração de temas e estilos padrões.

3. Descreva o funcionamento do sistema de Grid do Vuetify. Como ele se compara ao sistema tradicional de CSS Flexbox e Bootstrap?

O sistema de Grid do Vuetify é baseado no Flexbox e inspirado no grid system do Bootstrap. Ele organiza o layout em linhas (v-row) e colunas (v-col), permitindo criar interfaces responsivas de forma intuitiva. A vantagem é a integração nativa com os componentes Vuetify, evitando a necessidade de escrever muitas classes CSS personalizadas.

4. Quais são os principais componentes do Vuetify? Escolha pelo menos três e explique suas funcionalidades e usos.

- **v-btn:** botão altamente personalizável, usado para ações como salvar, enviar ou navegar.
- **v-card:** componente para exibir informações em blocos organizados, muito utilizado em dashboards e listagens.
- **v-dialog:** componente para criar caixas de diálogo modais, úteis para formulários rápidos, confirmações e mensagens.

5. Explique a diferença entre os componentes v-btn, v-card e v-dialog. Em quais situações cada um deles é mais adequado?

O v-btn é utilizado para ações diretas, como cliques em formulários ou navegação. O v-card é ideal para exibir informações agrupadas de forma organizada e visualmente agradável. Já o v-dialog é usado em situações em que se deseja exibir informações temporárias ou solicitar interações sem sair da página atual.

6. Como funciona a validação de formulários no Vuetify? Qual a importância do atributo rules ao definir um campo de entrada de dados?

A validação é feita utilizando o atributo rules, que recebe um array de funções responsáveis por validar os dados inseridos. Cada função retorna true quando a validação é bem-sucedida ou uma mensagem de erro quando falha. Esse sistema garante que apenas dados válidos sejam enviados, aumentando a confiabilidade da aplicação.

7. Explique como funcionam as Data Tables no Vuetify e como elas podem ser utilizadas para exibir grandes volumes de dados.

As Data Tables (v-data-table) permitem exibir dados em formato tabular, com suporte a paginação, ordenação e filtros. Elas são ideais para exibir grandes conjuntos de informações de forma organizada e interativa, podendo ser preenchidas dinamicamente com dados vindos de APIs.

8. O Vuetify permite a personalização avançada da interface. Explique como criar e configurar temas customizados.

A personalização de temas pode ser feita configurando o objeto theme ao inicializar o Vuetify no projeto. É possível definir cores primárias, secundárias, de erro, sucesso, entre outras. Além disso, pode-se alternar entre tema claro e escuro, ou até mesmo criar múltiplos temas para diferentes contextos da aplicação.

9. Qual é a importância do uso de SCSS no Vuetify? Como ele pode ser utilizado para modificar estilos globalmente?

O SCSS permite modificar variáveis de estilo globais, como cores e espaçamentos, antes mesmo da compilação do projeto. Isso garante maior flexibilidade e consistência

visual em toda a aplicação, já que mudanças podem ser aplicadas em larga escala de forma centralizada.

10. Como integrar APIs com Vuetify para preencher dinamicamente tabelas e formulários? Cite um exemplo de requisição e consumo de dados.

A integração pode ser feita utilizando bibliotecas como Axios ou Fetch API. Por exemplo, uma requisição GET pode buscar dados de uma API e preencher uma v-data-table. Da mesma forma, formulários podem ser preenchidos com dados vindos do servidor e enviados novamente através de requisições POST ou PUT.

Questões Práticas

1. Crie um projeto Vue.js com Vuetify e implemente um layout básico utilizando o sistema de Grid. O layout deve conter três colunas que se ajustam para telas pequenas.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Layout Vue + Vuetify</title>
  <script src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
  <link href="https://cdn.jsdelivr.net/npm/vuetify@2.6.16/dist/vuetify.min.css" rel="stylesheet">
  <script src="https://cdn.jsdelivr.net/npm/vuetify@2.6.16/dist/vuetify.js"></script>
</head>
<body>

  <div id="app">
    <v-app>
      <v-container>
        <v-row>
          <v-col cols="12" sm="4">
            <v-card class="pa-4" outlined>
              <h3>Coluna 1</h3>
              <p>Conteúdo da primeira coluna.</p>
            </v-card>
          </v-col>
          <v-col cols="12" sm="4">
            <v-card class="pa-4" outlined>
              <h3>Coluna 2</h3>
              <p>Conteúdo da segunda coluna.</p>
            </v-card>
          </v-col>
          <v-col cols="12" sm="4">
            <v-card class="pa-4" outlined>
              <h3>Coluna 3</h3>
              <p>Conteúdo da terceira coluna.</p>
            </v-card>
          </v-col>
        </v-row>
      </v-container>
    </v-app>
  </div>

  <script>
    new Vue({
      el: '#app',
      vuetify: new Vuetify(),
    })
  </script>

</body>
</html>
```

2. Crie um botão (v-btn) estilizado com uma cor personalizada e um evento de clique que exibe uma mensagem no console.

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Botão Vuetify</title>
    <link
      href="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.css"
      rel="stylesheet"
    />
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.js"></script>
  </head>
  <body>
    <div id="app">
      <v-app>
        <v-main class="d-flex align-center justify-center" style="height: 100vh;">
          <v-btn
            color="#9c27b0"
            class="text-white"
            @click="mostrarMensagem"
          >
            Clique em mim
          </v-btn>
        </v-main>
      </v-app>
    </div>
```

```html
    <script>
      const { createApp } = Vue;
      const { createVuetify } = Vuetify;
      const vuetify = createVuetify();
      createApp({
        methods: {
          mostrarMensagem() {
            console.log("Você clicou no botão personalizado!");
          }
        }
      }).use(vuetify).mount("#app");
    </script>
  </body>
</html>
```

3. Implemente um v-card contendo uma imagem, um título e um botão que exibe um alerta ao ser pressionado.

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Card Vuetify</title>
    <link
      href="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.css"
      rel="stylesheet"
    />
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.js"></script>
  </head>
  <body>
    <div id="app">
      <v-app>
        <v-main class="d-flex align-center justify-center" style="height: 100vh;">
          <v-card max-width="344">
            <v-img
              src="https://picsum.photos/400/200"
              height="200px"
            ></v-img>
            <v-card-title>Título do Card</v-card-title>
            <v-card-actions>
              <v-btn color="primary" @click="mostrarAlerta">Clique aqui</v-btn>
            </v-card-actions>
          </v-card>
        </v-main>
      </v-app>
    </div>
```

4. Crie um modal (v-dialog) que seja ativado ao clicar em um botão. O modal deve conter um título, uma mensagem e um botão para fechá-lo.

```html
<body>
  <div id="app">
    <v-app>
      <v-main class="d-flex align-center justify-center" style="height: 100vh;">
        <v-btn color="primary" @click="dialog = true">Abrir Modal</v-btn>
        <v-dialog v-model="dialog" max-width="400">
          <v-card>
            <v-card-title class="text-h6">Título do Modal</v-card-title>
            <v-card-text>Esta é a mensagem dentro do modal.</v-card-text>
            <v-card-actions>
              <v-btn color="primary" @click="dialog = false">Fechar</v-btn>
            </v-card-actions>
          </v-card>
        </v-dialog>
      </v-main>
    </v-app>
  </div>
```

```
    <script>
      const { createApp } = Vue
      const { createVuetify } = Vuetify

      const vuetify = createVuetify()

      createApp({
        data() {
          return {
            dialog: false
          }
        }
      }).use(vuetify).mount("#app")
    </script>
  </body>
</html>
```

5. Desenvolva um v-toolbar fixo na parte superior da tela que contenha um menu lateral (v-navigation-drawer) e um título centralizado.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Toolbar com Menu</title>
    <link
      href="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.css"
      rel="stylesheet"
    />
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.js"></script>
  </head>
  <body>
    <div id="app">
      <v-app>
        <v-navigation-drawer v-model="drawer" temporary>
          <v-list>
            <v-list-item title="Início"></v-list-item>
            <v-list-item title="Sobre"></v-list-item>
            <v-list-item title="Contato"></v-list-item>
          </v-list>
        </v-navigation-drawer>

        <v-app-bar app>
          <v-app-bar-nav-icon @click="drawer = !drawer"></v-app-bar-nav-icon>
          <v-toolbar-title class="text-center w-100">Meu Aplicativo</v-toolbar-title>
        </v-app-bar>

        <v-main class="pa-4">
          <p>Conteúdo da página aqui</p>
        </v-main>
      </v-app>
    </div>
```

```
    <script>
      const { createApp } = Vue
      const { createVuetify } = Vuetify

      const vuetify = createVuetify()

      createApp({
        data() {
          return {
            drawer: false
          }
        }
      }).use(vuetify).mount("#app")
    </script>
  </body>
</html>
```

6. Crie uma Data Table (v-data-table) que carregue uma lista de produtos a partir de um array estático no data(), contendo colunas para nome, preço e estoque.

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Data Table Produtos</title>
    <link
      href="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.css"
      rel="stylesheet"
    />
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.js"></script>
  </head>
  <body>
    <div id="app">
      <v-app>
        <v-main class="pa-4">
          <v-data-table
            :headers="headers"
            :items="produtos"
            class="elevation-1"
          ></v-data-table>
        </v-main>
      </v-app>
    </div>
```

```
    <script>
      const { createApp } = Vue
      const { createVuetify } = Vuetify

      const vuetify = createVuetify()

      createApp({
        data() {
          return {
            headers: [
              { text: "Nome", value: "nome" },
              { text: "Preço", value: "preco" },
              { text: "Estoque", value: "estoque" }
            ],
            produtos: [
              { nome: "Notebook", preco: "R$ 4.500", estoque: 12 },
              { nome: "Smartphone", preco: "R$ 2.000", estoque: 30 },
              { nome: "Monitor", preco: "R$ 800", estoque: 18 },
              { nome: "Teclado", preco: "R$ 200", estoque: 50 }
            ]
          }
        }
      }).use(vuetify).mount("#app")
    </script>
  </body>
</html>
```

7. Implemente um formulário de login utilizando Vuetify, com validação de campos para email e senha. Os campos devem exibir mensagens de erro quando não preenchidos corretamente.

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Login Vuetify</title>
    <link
      href="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.css"
      rel="stylesheet"
    />
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/vuetify@3.6.11/dist/vuetify.min.js"></script>
  </head>
  <body>
    <div id="app">
      <v-app>
        <v-main class="d-flex justify-center align-center" style="height: 100vh;">
          <v-card class="pa-6" max-width="400">
            <v-card-title class="text-h5 text-center">Login</v-card-title>
            <v-card-text>
              <v-form v-model="formValido" ref="form">
                <v-text-field
                  v-model="email"
                  label="Email"
                  type="email"
                  :rules="emailRules"
                  required
                ></v-text-field>
                <v-text-field
                  v-model="senha"
                  label="Senha"
                  type="password"
                  :rules="senhaRules"
                  required
                ></v-text-field>
                <v-btn
                  color="primary"
```

```
                block
                class="mt-4"
                @click="submitForm"
              >
                Entrar
              </v-btn>
            </v-form>
          </v-card-text>
        </v-card>
      </v-main>
    </v-app>
  </div>
  <script>
    const { createApp, ref } = Vue
    const { createVuetify } = Vuetify
    const vuetify = createVuetify()
    createApp({
      setup() {
        const formValido = ref(false)
        const form = ref(null)
        const email = ref("")
        const senha = ref("")
        const emailRules = [
          v => !!v || "O email é obrigatório",
          v => /.+@.+\..+/.test(v) || "E-mail inválido"
        ]
        const senhaRules = [
          v => !!v || "A senha é obrigatória",
          v => v.length >= 6 || "A senha deve ter pelo menos 6 caracteres"
        ]
        const submitForm = () => {
          if (form.value.validate()) {
            alert(`Login realizado!\nEmail: ${email.value}\nSenha: ${senha.value}`)
          }
        }
        return {
          formValido,
          form,
          email,
          senha,
          emailRules,
          senhaRules,
          submitForm
        }
      }
    }).use(vuetify).mount("#app")
  </script>
</body>
</html>
```

8. Crie um tema customizado no Vuetify, alterando a cor primária para verde e a cor secundária para amarelo. Aplique essas cores nos componentes da interface.

vuetify.js

```
import 'vuetify/styles'
import { createVuetify } from 'vuetify'
import { md3 } from 'vuetify/blueprints'

export const vuetify = createVuetify({
  blueprint: md3,
  theme: {
    defaultTheme: 'customTheme',
    themes: {
      customTheme: {
        dark: false,
        colors: {
          primary: '#4CAF50',
          secondary: '#FFEB3B',
          background: '#FFFFFF',
          surface: '#FFFFFF',
          error: '#B00020',
          info: '#2196F3',
          success: '#4CAF50',
          warning: '#FFC107',
        },
      },
    },
  },
})
```

Vue

```
<template>
  <v-container>
    <v-btn color="primary" class="ma-2">Botão Primário</v-btn>
    <v-btn color="secondary" class="ma-2">Botão Secundário</v-btn>

    <v-card class="ma-4" color="primary" dark>
      <v-card-title>Card com cor primária</v-card-title>
      <v-card-text>Este card usa a cor primária do tema.</v-card-text>
    </v-card>

    <v-chip color="secondary" class="ma-2">Chip Secundário</v-chip>
  </v-container>
</template>

<script setup lang="ts">
</script>

<style scoped>
</style>
```

9. Utilizando SCSS, modifique a aparência dos botões no Vuetify para que fiquem arredondados e com uma sombra suave.

```vue
<template>
  <v-container>
    <v-btn color="primary" class="ma-2">Botão Primário</v-btn>
    <v-btn color="secondary" class="ma-2">Botão Secundário</v-btn>

    <v-card class="ma-4" color="primary" dark>
      <v-card-title>Card com cor primária</v-card-title>
      <v-card-text>Este card usa a cor primária do tema.</v-card-text>
    </v-card>

    <v-chip color="secondary" class="ma-2">Chip Secundário</v-chip>
  </v-container>
</template>

<script setup lang="ts">
</script>

<style scoped lang="scss">
.v-btn {
  border-radius: 12px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.15);
  text-transform: none;
  transition: all 0.3s ease;
}

.v-btn:hover {
  box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2);
}
</style>
```

10. Faça a integração de uma API externa com Vuetify. Os dados da API devem ser exibidos dinamicamente em uma Data Table, atualizando os registros ao carregar a página.

```vue
<template>
  <v-container>
    <v-data-table
      :headers="headers"
      :items="users"
      :loading="loading"
      loading-text="Carregando dados..."
      class="elevation-1"
    >
    </v-data-table>
  </v-container>
</template>

<script setup lang="ts">
import { ref, onMounted } from 'vue'
import axios from 'axios'

interface User {
  id: number
  name: string
  username: string
  email: string
}

const users = ref<User[]>([])
const loading = ref(false)

const headers = [
  { text: 'ID', value: 'id' },
  { text: 'Nome', value: 'name' },
  { text: 'Username', value: 'username' },
  { text: 'Email', value: 'email' }
]

const fetchUsers = async () => {
  loading.value = true
  try {
    const response = await axios.get(
    'https://jsonplaceholder.typicode.com/users')
    users.value = response.data
  } catch (error) {
    console.error('Erro ao buscar dados da API:', error)
  } finally {
    loading.value = false
  }
}

onMounted(() => {
  fetchUsers()
})
</script>

<style scoped>
</style>
```