# ■ Guia de Bolso Vue.js – Super Detalhado (Básico → Avançado)

*Use este manual como referência rápida no dia a dia. Cada item traz descrição, uso contextualizado e um exemplo fictício para fixar.*

## Básico

### *data (Options API)*

Declara o estado reativo do componente (Options API).

➡■ Contexto (real): Formulário simples que exibe o nome digitado.

```
export default {
  data() {
    return { nome: '' }
  }
}
```

■ Exemplo fictício: Inicializa com um valor padrão.

```
export default {
  data() {
    return { contador: 1 }
  }
}
```

## Básico

### *methods*

Funções ligadas ao componente; úteis para eventos e lógicas acionáveis.

➡■ Contexto (real): Botão que salva os dados e exibe feedback.

```
export default {
  data(){ return { nome: '' } },
  methods: {
    salvar(){
      if(!this.nome) return alert('Digite o nome');
      // chamar API...
      alert('Salvo!');
    }
  }
}
```

■ Exemplo fictício: Incrementar um contador ao clicar.

```
export default {
  data(){ return { n: 0 } },
  methods: { inc(){ this.n++ } }
}
```

## Básico

### *computed*

Propriedades derivadas com cache automático baseadas em dependências reativas.

➡■ Contexto (real): Total do carrinho calculado a partir dos itens.

```
export default {
```

```
  data(){ return { itens:[{qtd:2, preco:10},{qtd:1, preco:7}] } },
  computed:{
    total(){ return this.itens.reduce((s,i)=>s+i.qtd*i.preco,0) }
  }
}
```

■ Exemplo fictício: Saudação com nome completo a partir de nome e sobrenome.

```
export default {
  data(){ return { nome:'Ana', sobrenome:'Silva' } },
  computed:{ saudacao(){ return `Olá, ${this.nome} ${this.sobrenome}` } }
}
```

## Básico

### *watch*

Observa mudanças e executa efeitos colaterais controlados.

➡■ Contexto (real): Buscar produtos sempre que o filtro de categoria mudar.

```
export default {
  data(){ return { categoria:'tudo' } },
  watch:{
    categoria(nova){ this.buscarProdutos(nova) }
  },
  methods:{ buscarProdutos(cat){ /* chama API */ } }
}
```

■ Exemplo fictício: Logar quando a idade muda.

```
export default {
  data(){ return { idade: 18 } },
  watch:{ idade(n, a){ console.log('idade', a, '→', n) } }
}
```

## Básico

### *v-model*

Ligação bidirecional entre estado e inputs.

➡■ Contexto (real): Formulário que atualiza estado conforme digitação.

```
<input v-model="email" placeholder="Seu email" />
<script>
export default{
  data(){ return { email:'' } }
}
</script>
```

■ Exemplo fictício: Toggle simples com checkbox.

```
<label><input type="checkbox" v-model="aceito" /> Aceito</label>
<script>
export default{ data(){ return { aceito:false } } }
</script>
```

## Básico

### *v-bind / v-on*

Liga atributos dinamicamente (v-bind) e escuta eventos (v-on ou @).

➡■ Contexto (real): Desabilitar botão enquanto carrega e tratar click.

```
<button :disabled="carregando" @click="salvar">Salvar</button>
```

```
<script>
export default{ data(){return{carregando:false}}, methods:{ salvar(){/*...*/} } }
</script>
```

■ Exemplo fictício: Alterar classe dinamicamente e logar clique.

```
<button :class="{ ativo:isAtivo }" @click="isAtivo=!isAtivo">Toggle</button>
<script>
export default{ data(){return{isAtivo:false}} }
</script>
```

## Renderização

### *v-if / v-else / v-else-if*

Renderização condicional (remove/adiciona no DOM).

➡■ Contexto (real): Mostrar botão de login ou logout.

```
<button v-if="logado" @click="sair">Sair</button>
<button v-else @click="entrar">Entrar</button>
```

■ Exemplo fictício: Exibir aviso apenas quando houver erro.

```
<p v-if="erro">Ocorreu um erro. Tente novamente.</p>
```

## Renderização

### *v-show*

Mostra/esconde via CSS (display), sem remover do DOM.

➡■ Contexto (real): Menu que expande/contrai sem recriar conteúdo.

```
<nav v-show="aberto"> ... </nav>
```

■ Exemplo fictício: Mostrar dica apenas quando hover (controlado por boolean).

```
<div @mouseover="show=true" @mouseleave="show=false">
  <span v-show="show">Dica!</span>
</div>
```

## Renderização

### *v-for + key*

Itera sobre listas; 'key' ajuda o Vue a rastrear nós.

➡■ Contexto (real): Renderizar itens do carrinho.

```
<li v-for="item in itens" :key="item.id">
  {{ item.nome }} - {{ item.preco }}
</li>
```

■ Exemplo fictício: Listar tarefas com índice como key (não recomendado, mas ilustrativo).

```
<li v-for="(t, i) in tarefas" :key="i">{{ t }}</li>
```

## Renderização

### *Vinculação de classes e estilos*

Controle condicional de classes/estilos inline.

➡■ Contexto (real): Aplicar classe 'invalido' quando houver erro.

```
<input :class="{ invalido: temErro }" />
```

■ Exemplo fictício: Mudar cor via style dinâmico.

```
<div :style="{ color: corPreferida }">Texto</div>
```

## Componentes

### *props (tipos, default, validação)*

Receber dados do pai com tipos/validação.

➡■ Contexto (real): Card que recebe título (string) e destacado (boolean) com default.

```
export default {
  props:{
    titulo:{ type:String, required:true },
    destacado:{ type:Boolean, default:false }
  }
}
```

■ Exemplo fictício: Avatar que recebe tamanho com validação customizada.

```
export default {
  props:{
    size:{
      type:Number,
      default:40,
      validator:v=>[24,40,80].includes(v)
    }
  }
}
```

## Componentes

### *emit (Options) / defineEmits (Composition)*

Filho comunica eventos ao pai.

➡■ Contexto (real): Modal emite 'fechar' quando clicar Sair.

```
// Filho
this.$emit('fechar')
// Pai
<Modal @fechar="aberto=false" />
```

■ Exemplo fictício: Botão contador emite 'update' com novo valor.

```
// Filho
this.$emit('update', this.count)
// Pai
<BtnCounter @update="n => total = n" />
```

## Componentes

### *slots (default, nomeados e scoped)*

Permitir conteúdo customizável pelo pai.

➡■ Contexto (real): Card com slots de header e footer.

```
<Card>
  <template #header> Título </template>
  Conteúdo do card
  <template #footer> Rodapé </template>
</Card>
```

■ Exemplo fictício: Slot com escopo fornecendo 'item' para o pai renderizar.

```
<Lista :itens="itens">
  <template #default="{ item }">
    <b>{{ item.nome }}</b>
  </template>
</Lista>
```

## Componentes

### *provide/inject*

Compartilhar dados/serviços sem prop drilling.

➡■ Contexto (real): Tema fornecido pelo layout para componentes internos.

```
// Pai
provide('tema', ref('claro'))
// Filho
const tema = inject('tema')
```

■ Exemplo fictício: Serviço de i18n simples.

```
// provide('t', (k)=>dict[k] )
// inject('t')('hello')
```

## Componentes

### *component is / componentes dinâmicos*

Trocar o componente renderizado dinamicamente.

➡■ Contexto (real): Alternar entre visualizações de lista e grade.

```
<component :is="modo === 'grid' ? GridView : ListView" />
```

■ Exemplo fictício: Wizard de etapas com componente dinâmico.

```
<component :is="passos[etapaAtual]" />
```

## Componentes

### *Componentes assíncronos*

Carregar componentes sob demanda (code-splitting).

➡■ Contexto (real): Página pesada carregada apenas quando acessada.

```
const Pesado = defineAsyncComponent(()=>import('./Pesado.vue'))
```

■ Exemplo fictício: Modal carregado sob demanda.

```
const Modal = defineAsyncComponent(()=>import('./Modal.vue'))
```

## Reatividade (Composition API)

### *ref*

Valor primitivo/objeto reativo com .value.

➡■ Contexto (real): Contador com ref e botão de incremento.

```
import { ref } from 'vue'
const n = ref(0)
function inc(){ n.value++ }
```

■ Exemplo fictício: Flag booleana de loading.

```
const loading = ref(false)
```

## Reatividade (Composition API)

### *reactive / toRefs*

Objetos reativos (proxy). toRefs expõe props reativas individuais.

➡■ Contexto (real): Formulário com reactive e destruturação segura.

```
const form = reactive({ email:'', senha:'' })
const { email, senha } = toRefs(form)
```

■ Exemplo fictício: Config com nested state.

```
const cfg = reactive({ tema:{ cor:'blue' } })
```

## Reatividade (Composition API)

### *computed (getter/setter)*

Derivar valores e também escrever (2-way).

➡■ Contexto (real): Campo com máscara que grava limpo.

```
const bruto = ref('')
const valor = computed({
  get:()=>formatar(bruto.value),
  set:(v)=>{ bruto.value = limpar(v) }
})
```

■ Exemplo fictício: Nome completo com set atualizando partes.

```
const nome = ref('Ana'), sobrenome=ref('Silva')
const completo = computed({
  get:()=>`${nome.value} ${sobrenome.value}`,
  set:(v)=>{ [nome.value, sobrenome.value] = v.split(' ') }
})
```

## Reatividade (Composition API)

### *watch / watchEffect*

watch: dependências declaradas. watchEffect: executa e rastreia automaticamente.

➡■ Contexto (real): Debounce de busca ao digitar.

```
watch(busca, debounce((q)=>carregar(q), 300))
```

■ Exemplo fictício: Logar quaisquer dependências que mudarem.

```
watchEffect(()=>{
  console.log('total:', total.value, 'itens:', itens.value.length)
})
```

## Reatividade (Composition API)

### *nextTick*

Executar após o DOM atualizar.

➡■ Contexto (real): Scroll automático para o final da lista após adicionar item.

```
await nextTick()
container.scrollTop = container.scrollHeight
```

■ Exemplo fictício: Focar input depois de mostrar modal.

```
await nextTick(); inputRef.value.focus()
```

## Modificadores e Diretrizes

### *v-on modifiers (stop, prevent, once, capture, passive)*

Controlar propagação/comportamento de eventos.

➡■ Contexto (real): Form que não recarrega a página.

```
<form @submit.prevent="salvar">...</form>
```

■ Exemplo fictício: Clique que ocorre apenas uma vez.

```
<button @click.once="iniciar">Começar</button>
```

## Modificadores e Diretrizes

### *v-model modifiers (trim, number, lazy)*

Ajustar comportamento de v-model.

➡■ Contexto (real): Converter string numérica para número.

```
<input v-model.number="idade" type="number" />
```

■ Exemplo fictício: Remover espaços nas pontas.

```
<input v-model.trim="nome" />
```

## Modificadores e Diretrizes

### *Diretivas personalizadas*

Criar comportamentos reutilizáveis no DOM.

➡■ Contexto (real): Auto-focus em inputs.

```
app.directive('focus', { mounted(el){ el.focus() } })
```

■ Exemplo fictício: Máscara simplificada.

```
app.directive('upper', { update(el){ el.value = el.value.toUpperCase() } })
```

## Router (vue-router)

### *Rotas básicas, params e navegação programática*

Definir rotas, acessar params e navegar via código.

➡■ Contexto (real): Perfil com :id e botão voltar.

```
const routes=[{ path:'/perfil/:id', component: Perfil }]
router.push({ name:'perfil', params:{ id:42 } })
```

■ Exemplo fictício: Ler query string.

```
// /busca?q=vue
const route = useRoute()
console.log(route.query.q)
```

## Router (vue-router)

### *Rotas aninhadas e lazy-loading*

Layout com aninhado e carregamento dinâmico.

➡■ Contexto (real): Área logada com subrotas.

```
const routes=[{
  path:'/app',
```

```
  component:()=>import('./Layout.vue'),
  children:[
    { path:'home', component:()=>import('./Home.vue') }
  ]
}]
```

■ Exemplo fictício: Página About carregada quando acessada.

```
{ path:'/about', component:()=>import('./About.vue') }
```

## Router (vue-router)

### *Guards de navegação (beforeEach, beforeEnter)*

Proteger rotas privadas checando auth.

➡■ Contexto (real): Redirecionar se não logado.

```
router.beforeEach((to,from,next)=>{
  if(to.meta.privada && !store.isAuth) next('/login')
  else next()
})
```

■ Exemplo fictício: Guard por rota.

```
{ path:'/admin', component:Admin, beforeEnter:()=>checarAdmin() }
```

## Estado Global (Pinia/Vuex)

### *Pinia – defineStore, state, actions, getters*

Gerenciar carrinho de compras.

➡■ Contexto (real): Store de carrinho com actions.

```
export const useCart = defineStore('cart',{
  state:()=>({ itens:[] }),
  getters:{ total:(s)=>s.itens.reduce((t,i)=>t+i.preco*i.qtd,0) },
  actions:{ add(p){ this.itens.push(p) } }
})
```

■ Exemplo fictício: Contador global simples.

```
export const useCount = defineStore('count',{
  state:()=>({ n:0 }), actions:{ inc(){ this.n++ } }
})
```

## Estado Global (Pinia/Vuex)

### *Persistência simples (localStorage)*

Salvar e restaurar estado do usuário.

➡■ Contexto (real): Sincronizar store com localStorage.

```
watch(()=>store.$state, (s)=>{
  localStorage.setItem('app', JSON.stringify(s))
},{ deep:true })
```

■ Exemplo fictício: Carregar estado inicial.

```
store.$patch(JSON.parse(localStorage.getItem('app') || '{}'))
```

## APIs & HTTP

### *axios básico + loading/erro*

Carregar lista com estado de loading e erro.

➡■ Contexto (real): Listar usuários com try/catch.

```
const usuarios = ref([]), loading=ref(false), erro=ref('')
async function carregar(){
  try{
    loading.value=true
    usuarios.value = (await axios.get('/api/users')).data
  }catch(e){ erro.value='Falha ao carregar' }
  finally{ loading.value=false }
}
```

■ Exemplo fictício: Criar recurso com POST.

```
await axios.post('/api/items', { nome: 'Item X' })
```

## APIs & HTTP

### *Interceptors, headers e cancelamento*

Injetar token JWT e cancelar requisições antigas.

➡■ Contexto (real): Interceptor de auth + AbortController.

```
axios.interceptors.request.use(cfg=>{
  cfg.headers.Authorization = 'Bearer ' + token
  return cfg
})

const ctrl = new AbortController()
axios.get('/api/busca', { signal: ctrl.signal })
ctrl.abort()
```

■ Exemplo fictício: Retry simples.

```
try{ await axios.get('/api') }catch(e){ await axios.get('/api') }
```

## Formulários & Validação

### *Validação manual e feedback*

Exibir erros abaixo dos campos.

➡■ Contexto (real): Requerer email e senha.

```
const erros = reactive({})
function validar(){
  erros.email = !email.value ? 'Obrigatório' : ''
  erros.senha = senha.value.length<6 ? 'Mínimo 6' : ''
}
```

■ Exemplo fictício: Desabilitar submit se houver erros.

```
<button :disabled="Object.values(erros).some(Boolean)">Salvar</button>
```

## Transições

### *e*

Animar entrada/saída de elementos e listas.

➡■ Contexto (real): Fade simples.

```
<transition name="fade">
  <p v-if="mostrar">Olá</p>
</transition>
<style>
```

```
.fade-enter-active,.fade-leave-active{ transition: opacity .2s }
.fade-enter-from,.fade-leave-to{ opacity:0 }
</style>
```

■ Exemplo fictício: Lista animada.

```
<transition-group name="list" tag="ul">
  <li v-for="i in itens" :key="i.id">{{ i.nome }}</li>
</transition-group>
```

## Performance & Otimizações

### *keep-alive, v-once, memoização*

Cachear abas de rotas e evitar recomputes caros.

➡■ Contexto (real): Cache de views com keep-alive.

```
<keep-alive><router-view /></keep-alive>
```

■ Exemplo fictício: Renderizar estático com v-once.

```
<h1 v-once>Título que não muda</h1>
```

## Erros & Debug

### *errorCaptured, Vue Devtools*

Capturar erros de filhos e inspecionar estado.

➡■ Contexto (real): Boundary simples.

```
errorCaptured(err, vm, info){
  console.error('Erro capturado:', err, info); return false
}
```

■ Exemplo fictício: Usar Devtools para inspecionar componentes/estado.

```
// Instale a extensão Vue Devtools para seu navegador
```

## Segurança

### *XSS e v-html*

Evite renderizar HTML arbitrário sem sanitização.

➡■ Contexto (real): Sanitizar antes de usar v-html.

```
<div v-html="htmlSeguro"></div> // nunca injete input do usuário cru
```

■ Exemplo fictício: Escapar conteúdo em vez de usar v-html.

```
<div>{{ conteudo }}</div>
```

## Internacionalização & Acessibilidade

### *vue-i18n + atributos ARIA*

Trocar idioma e adicionar aria-labels.

➡■ Contexto (real): Uso básico do i18n.

```
const { t, locale } = useI18n()
locale.value = 'pt-BR'
<p>{{ t('hello') }}</p>
```

■ Exemplo fictício: Botão com aria-label descritivo.

```
<button aria-label="Abrir carrinho">■</button>
```

## Ferramentas

### *ESLint + Prettier*

Padronizar código e evitar erros comuns.

➡■ Contexto (real): Script no package.json.

```
{
  "scripts": { "lint": "eslint --ext .js,.vue src", "format": "prettier -w ." }
}
```

■ Exemplo fictício: Regra para Vue 3.

```
// .eslintrc.cjs -> extends: ['plugin:vue/vue3-recommended']
```

## Ferramentas

### *Variáveis de ambiente*

Configurar bases de API separadas por ambiente.

➡■ Contexto (real): Vite ou Vue CLI usam arquivos .env.*.

```
// .env.development
VITE_API=http://localhost:3000
```

■ Exemplo fictício: Acessar a env no código.

```
const api = import.meta.env.VITE_API
```

## Padrões & Arquitetura

### *Estrutura de pastas, props down / events up*

Organizar o projeto e manter dados fluindo do pai para o filho.

➡■ Contexto (real): Estrutura comum.

```
src/
  api/
  components/
  pages/
  stores/
  composables/
  router/
  assets/
```

■ Exemplo fictício: Comunicação: pai → filho (props), filho → pai (emit).

```
<Filho :valor="x" @mudar="x=$event" />
```

## Composables úteis

### *useFetch, useDebounce, useLocalStorage*

Reutilizar lógica de dados e utilitários.

➡■ Contexto (real): useLocalStorage (exemplo).

```
export function useLocalStorage(key, init){
  const v = ref(JSON.parse(localStorage.getItem(key)) ?? init)
  watch(v, nv=>localStorage.setItem(key, JSON.stringify(nv)), { deep:true })
  return v
}
```

■ Exemplo fictício: useDebounce (exemplo simples).

```
export const useDebounce = (fn, t=300)=>{
  let id; return (...args)=>{ clearTimeout(id); id=setTimeout(()=>fn(...args),t) }
}
```

## ■ Checklist de Prática (marque ao avançar)

- Criei um CRUD completo (listar, criar, editar, excluir)?
- Usei v-model com modificadores (trim/number/lazy)?
- Implementei props com tipos/validação e emits tipados?
- Usei slots nomeados e scoped slots em um componente genérico?
- Implementei provide/inject para um serviço global simples?
- Implementei rotas dinâmicas, aninhadas e guards?
- Criei uma store Pinia com getters e actions; persisti no localStorage?
- Consumi API com axios usando interceptors, loading e tratamento de erro?
- Validei formulários e exibi feedback de erro?
- Implementei uma animação com e ?
- Otimizei com keep-alive e v-once onde cabia?
- Configurei ESLint + Prettier e scripts de lint/format?
- Evitei v-html ou sanitizei o conteúdo quando necessário?
- Usei Vue Devtools para depurar um problema real?
- Criei pelo menos 2 composables reutilizáveis?