Vue.js

Panduan Lengkap untuk Pemula & Menengah

■ Bahasa Indonesia

■ Vue 3 + Composition API

■■ Contoh Kode Praktis

■■ Dari Dasar hingga Mahir

Dibuat untuk Developer Indonesia

Vue.js 3.x | 2024

Daftar Isi

Bab 10 — Tips & Best Practices

- Performa
- Konvensi Kode
- Tools Berguna

Bab 1 — Pengenalan Vue.js

1.1 Apa itu Vue.js?

Vue.js adalah framework JavaScript progresif yang digunakan untuk membangun antarmuka pengguna (UI). Diciptakan oleh **Evan You** pada tahun 2014, Vue dirancang agar mudah dipelajari namun cukup powerful untuk membangun aplikasi skala besar. Vue dapat diintegrasikan ke dalam proyek yang sudah ada secara bertahap (progressive), artinya kamu bisa menggunakannya hanya untuk bagian tertentu dari halaman web.

1.2 Keunggulan Vue.js

■ **Mudah Dipelajari**	Sintaks yang bersih dan dokumentasi yang sangat baik membuat Vue ideal untuk pemula.
■ **Reaktivitas Otomatis**	Sistem reaktivitas Vue memperbarui UI secara otomatis saat data berubah.
■ **Progresif**	Bisa dipakai sebagian kecil atau sebagai full SPA framework.
■ **Ekosistem Lengkap**	Vue Router, Pinia, Vite, dan banyak library siap pakai.
■ **Performa Tinggi**	Virtual DOM yang dioptimalkan membuat rendering sangat cepat.
■ **Komunitas Besar**	Komunitas aktif dan banyak sumber belajar tersedia.

1.3 Perbandingan Framework

Fitur	Vue.js	React	Angular
Kurva Belajar	■ **Mudah**	■■ Sedang	■■■ Sulit
Bahasa	**JS / TS**	JSX / TS	TypeScript
State Management	**Pinia**	Redux / Zustand	NgRx / Services
Routing	**Vue Router**	React Router	Angular Router
Ukuran Bundle	**~20KB**	~30KB	~60KB
Perusahaan Utama	**Komunitas**	Meta	Google

Bab 2 — Instalasi & Setup

2.1 Menggunakan CDN (Cara Tercepat)

Cara paling cepat mencoba Vue adalah dengan menambahkan script CDN ke HTML.

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Vue.js Pertamaku</title>
</head>
<body>
<div id="app">
<h1>{{ pesan }}</h1>
<button @click="ubahPesan">Klik Saya!</button>
</div>

<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<script>
Vue.createApp({
data() {
return { pesan: 'Halo, Vue.js!' }
},
methods: {
ubahPesan() {
this.pesan = 'Vue.js itu Keren!'
}
}
}).mount('#app')
</script>
</body>
</html>
```

2.2 Membuat Proyek dengan Vite (Cara Profesional)

Untuk proyek sungguhan, gunakan **Vite** sebagai build tool. Vite sangat cepat dan sudah mendukung Hot Module Replacement (HMR).

Terminal

```
# Pastikan Node.js sudah terinstall (versi 16+)
npm create vue@latest nama-proyek-ku

# Masuk ke folder proyek
cd nama-proyek-ku

# Install dependencies
npm install

# Jalankan development server
npm run dev
```

■ **TIP:** Saat menjalankan **npm create vue@latest**, kamu akan ditanya beberapa pilihan seperti TypeScript, Vue Router, Pinia, dll. Untuk pemula, jawab "No" dulu untuk semua.

2.3 Struktur Folder Proyek

Folder/File	Keterangan
`src/`	Folder utama berisi kode aplikasi
`src/main.js`	Entry point aplikasi Vue
`src/App.vue`	Komponen root / utama
`src/components/`	Folder untuk komponen-komponen
`src/assets/`	Gambar, CSS, font
`src/views/`	Halaman-halaman (jika pakai Router)
`public/`	File statis yang tidak diproses
`package.json`	Daftar dependensi dan scripts
`vite.config.js`	Konfigurasi Vite

2.4 Anatomi File .vue (Single File Component)

Setiap komponen Vue disimpan dalam file **.vue** yang berisi tiga bagian utama:

```
KartuKomponen.vue
<template>
<!-- HTML template komponen -->
<div class="kartu">
<h2>{{ judul }}</h2>
<p>{{ deskripsi }}</p>
</div>
```

```
</template>

<script setup>
// JavaScript / logic komponen (Composition API)
import { ref } from 'vue'

const judul = ref('Judul Kartu')
const deskripsi = ref('Ini adalah deskripsi kartu.')
</script>

<style scoped>
/* CSS hanya berlaku di komponen ini */
.kartu {
padding: 16px;
border-radius: 8px;
background: #f0fdf8;
}
</style>
```

Bab 3 — Sintaks Template

3.1 Interpolasi Data (Mustache Syntax)

Gunakan kurung kurawal ganda **{{ }}** untuk menampilkan data di template.

Interpolasi.vue

```
<template>
<div>
<p>Nama: {{ nama }}</p>
<p>Umur: {{ umur }} tahun</p>
<p>Tahun lahir: {{ 2024 - umur }}</p>
<p>{{ pesan.toUpperCase() }}</p>
</div>
</template>

<script setup>
import { ref } from 'vue'

const nama = ref('Budi Santoso')
const umur = ref(25)
const pesan = ref('selamat datang di vue!')
</script>
```

3.2 Binding Atribut dengan v-bind

Untuk mengikat data ke atribut HTML, gunakan direktif **v-bind:** atau singkatannya **:**

Binding.vue

```
<template>
<!-- Cara panjang -->
<img v-bind:src="urlGambar" v-bind:alt="deskripsi">

<!-- Cara singkat (direkomendasikan) -->
<img :src="urlGambar" :alt="deskripsi">
<a :href="linkUrl">Kunjungi Website</a>
<button :disabled="sedangLoading">Simpan</button>
<div :class="{ aktif: isAktif, merah: adaError }">Kotak</div>
</template>

<script setup>
import { ref } from 'vue'
```

```
const urlGambar = ref('https://vuejs.org/logo.svg')
const deskripsi = ref('Logo Vue.js')
const linkUrl = ref('https://vuejs.org')
const sedangLoading = ref(false)
const isAktif = ref(true)
const adaError = ref(false)
</script>
```

3.3 Event Handling dengan v-on

Gunakan **v-on:** atau singkatannya @ untuk mendengarkan event dari elemen HTML.

```
EventHandling.vue
<template>
<div>
<p>Jumlah klik: {{ hitungan }}</p>

<!-- Cara panjang -->
<button v-on:click="tambah">Tambah</button>

<!-- Cara singkat (direkomendasikan) -->
<button @click="tambah">Tambah</button>
<button @click="kurang">Kurang</button>

<!-- Inline expression -->
<button @click="hitungan = 0">Reset</button>

<!-- Event Modifiers -->
<form @submit.prevent="kirimForm">
<input @keyup.enter="cariData">
<button type="submit">Kirim</button>
</form>
</div>
</template>

<script setup>
import { ref } from 'vue'

const hitungan = ref(0)

function tambah() { hitungan.value++ }
function kurang() { hitungan.value-- }
function kirimForm() { console.log('Form dikirim!') }
function cariData() { console.log('Mencari...') }
</script>
```

■ **TIP: Event Modifiers yang sering dipakai:**

- **.prevent** → mencegah default behavior (misal submit form refresh halaman)
- **.stop** → menghentikan event propagation
- **.once** → event hanya dijalankan sekali
- **.enter, .space, .esc** → key modifiers untuk keyboard

Bab 4 — Komponen Vue

Komponen adalah blok bangunan utama aplikasi Vue. Setiap komponen adalah file .vue yang berdiri sendiri dan dapat digunakan ulang di seluruh aplikasi.

4.1 Membuat & Menggunakan Komponen

KartuPengguna.vue

```
<!-- File: src/components/KartuPengguna.vue -->
<template>
<div class="kartu">
<img :src="foto" :alt="nama">
<h3>{{ nama }}</h3>
<p>{{ jabatan }}</p>
</div>
</template>

<script setup>
// Definisikan props yang diterima komponen ini
const props = defineProps({
nama: String,
jabatan: String,
foto: String
})
</script>

<style scoped>
.kartu { border: 1px solid #ddd; padding: 16px; border-radius: 8px; }
</style>
```

App.vue

```
<!-- File: src/App.vue — menggunakan komponen di atas -->
<template>
<div>
<h1>Tim Kami</h1>
<KartuPengguna
nama="Budi Santoso"
jabatan="Frontend Developer"
foto="https://i.pravatar.cc/100?img=1"
/>
<KartuPengguna
```

```
nama="Siti Rahayu"
jabatan="UI/UX Designer"
foto="https://i.pravatar.cc/100?img=2"
/>
</div>
</template>

<script setup>
import KartuPengguna from './components/KartuPengguna.vue'
</script>
```

4.2 Props — Mengirim Data ke Komponen Anak

Props adalah cara mengirim data dari komponen induk ke komponen anak. Props bersifat **one-way data flow** (satu arah dari atas ke bawah).

Produk.vue

```
<script setup>
// Definisi props dengan validasi tipe
const props = defineProps({
judul: {
type: String,
required: true
},
harga: {
type: Number,
default: 0
},
tersedia: {
type: Boolean,
default: true
},
tags: {
type: Array,
default: () => []
}
})
</script>

<template>
<div>
<h2>{{ props.judul }}</h2>
```

```
<p>Harga: Rp {{ props.harga.toLocaleString('id-ID') }}</p>
<span v-if="props.tersedia">Tersedia</span>
<span v-else>Habis</span>
</div>
</template>
```

4.3 Emit — Mengirim Event ke Komponen Induk

Komponen anak berkomunikasi ke induk dengan cara **emit** (memancarkan) event.

FormInput.vue (Komponen Anak)

```
<!-- Komponen Anak: FormInput.vue -->
<template>
<div>
<input v-model="nilaiInput" placeholder="Ketik sesuatu...">
<button @click="kirimData">Kirim ke Induk</button>
</div>
</template>

<script setup>
import { ref } from 'vue'

const nilaiInput = ref('')
const emit = defineEmits(['dataKirim'])

function kirimData() {
emit('dataKirim', nilaiInput.value)
nilaiInput.value = ''
}
</script>
```

App.vue (Komponen Induk)

```
<!-- Komponen Induk: App.vue -->
<template>
<div>
<FormInput @dataKirim="terimaData" />
<p>Data diterima: {{ dataYangDiterima }}</p>
</div>
</template>

<script setup>
import { ref } from 'vue'
import FormInput from './components/FormInput.vue'

const dataYangDiterima = ref('')
```

```
function terimaData(nilai) {
dataYangDiterima.value = nilai
}
</script>
```

4.4 Slots — Konten Dinamis

Slot memungkinkan komponen induk menyisipkan konten HTML ke dalam komponen anak.

KartuLayout.vue

```
<!-- KartuLayout.vue — komponen dengan slot -->
<template>
<div class="kartu">
<div class="header">
<slot name="header">Judul Default</slot>
</div>
<div class="konten">
<slot></slot> <!-- slot utama/default -->
</div>
<div class="footer">
<slot name="footer"></slot>
</div>
</div>
</template>
```

App.vue

```
<!-- Penggunaan slot di komponen induk -->
<KartuLayout>
<template #header>
<h2>Judul Kustom Saya</h2>
</template>

<!-- Konten untuk slot default -->
<p>Ini adalah isi kartu yang bisa berisi apapun!</p>

<template #footer>
<button>Tutup</button>
</template>
</KartuLayout>
```

Bab 5 — Reaktivitas Vue

5.1 ref() — Reaktivitas untuk Nilai Primitif

ref() digunakan untuk membuat nilai reaktif. Gunakan **.value** untuk mengakses atau mengubah nilainya di dalam script, tapi di template tidak perlu .value.

RefContoh.vue

```
<script setup>
import { ref } from 'vue'

// ref untuk nilai primitif (string, number, boolean)
const nama = ref('Budi')
const umur = ref(25)
const aktif = ref(true)

// Mengubah nilai: gunakan .value di dalam script
function ubahNama() {
nama.value = 'Siti'
umur.value++
aktif.value = !aktif.value
}

// Array dan Object juga bisa menggunakan ref
const daftarNama = ref(['Budi', 'Siti', 'Andi'])
function tambahNama() {
daftarNama.value.push('Baru')
}
</script>

<template>
<!-- Di template, tidak perlu .value -->
<p>{{ nama }} ({{ umur }} tahun)</p>
<p>Status: {{ aktif ? 'Aktif' : 'Tidak Aktif' }}</p>
<ul>
<li v-for="n in daftarNama" :key="n">{{ n }}</li>
</ul>
</template>
```

5.2 reactive() — Reaktivitas untuk Object

reactive() digunakan untuk membuat objek reaktif. Berbeda dengan ref(), tidak perlu .value.

ReactiveContoh.vue

```
<script setup>
import { reactive } from 'vue'

// Reaktif untuk objek
const pengguna = reactive({
nama: 'Budi Santoso',
email: 'budi@email.com',
alamat: {
kota: 'Surabaya',
provinsi: 'Jawa Timur'
}
})

// Modifikasi langsung tanpa .value
function ubahKota(kotaBaru) {
pengguna.alamat.kota = kotaBaru
}
</script>

<template>
<p>{{ pengguna.nama }} — {{ pengguna.alamat.kota }}</p>
<button @click="ubahKota('Jakarta')">Pindah ke Jakarta</button>
</template>
```

Aspek	ref()	reactive()
Untuk	Primitif & Object	Object/Array saja
Akses di script	.value diperlukan	Langsung akses
Akses di template	Otomatis unwrap	Langsung akses
Destructuring	Aman	Kehilangan reaktivitas

5.3 Computed Properties

computed() adalah nilai yang dihitung secara otomatis berdasarkan data reaktif lain. Hasilnya di-cache dan hanya diperbarui ketika dependensinya berubah.

Computed.vue

```
<script setup>
import { ref, computed } from 'vue'

const harga = ref(100000)
const jumlah = ref(3)
```

```
const diskon = ref(10) // persen

// Computed: dihitung otomatis dari harga & jumlah
const subtotal = computed(() => harga.value * jumlah.value)

// Computed dengan logika lebih kompleks
const total = computed(() => {
const sub = subtotal.value
const potongan = sub * (diskon.value / 100)
return sub - potongan
})

const pesanTotal = computed(() =>
`Total: Rp ${total.value.toLocaleString('id-ID')}`
)
</script>

<template>
<div>
<p>Harga satuan: Rp {{ harga.toLocaleString('id-ID') }}</p>
<p>Jumlah: {{ jumlah }}</p>
<p>Subtotal: Rp {{ subtotal.toLocaleString('id-ID') }}</p>
<p><b>{{ pesanTotal }}</b></p>
</div>
</template>
```

5.4 Watchers

watch() digunakan untuk memantau perubahan data reaktif dan menjalankan fungsi sebagai respons.

Watcher.vue

```
<script setup>
import { ref, watch, watchEffect } from 'vue'

const query = ref('')
const hasil = ref([])

// watch: pantau satu nilai
watch(query, (nilaiBaru, nilaiLama) => {
console.log(`Query berubah: ${nilaiLama} → ${nilaiBaru}`)
// Misalnya: cari data dari API
if (nilaiBaru.length > 2) {
cariData(nilaiBaru)
}
```

```
})

// watchEffect: otomatis pantau semua dependensi
watchEffect(() => {
// Dijalankan segera, lalu setiap kali query berubah
console.log('Query sekarang:', query.value)
})

// watch dengan immediate (langsung dijalankan saat komponen mount)
watch(query, (val) => {
console.log(val)
}, { immediate: true, deep: true })
</script>
```

Bab 6 — Composition API

Composition API adalah cara modern menulis komponen Vue 3. Menggunakan fungsi-fungsi seperti **ref()**, **computed()**, dan **watch()** di dalam blok **<script setup>**.

6.1 Lifecycle Hooks

Lifecycle hooks memungkinkan kamu menjalankan kode pada tahap tertentu dalam siklus hidup komponen.

Hook	Kapan Dijalankan
`onMounted`	Setelah komponen dipasang ke DOM
`onUpdated`	Setelah data berubah dan DOM diperbarui
`onUnmounted`	Sebelum komponen dihapus dari DOM
`onBeforeMount`	Sebelum komponen dipasang ke DOM
`onBeforeUpdate`	Sebelum DOM diperbarui
`onBeforeUnmount`	Sebelum komponen dihapus
`onErrorCaptured`	Ketika ada error dari komponen anak

LifecycleHooks.vue

```
<script setup>
import { ref, onMounted, onUnmounted, onUpdated } from 'vue'

const data = ref(null)
const loading = ref(true)

onMounted(async () => {
// Dijalankan saat komponen siap
console.log('Komponen sudah mounted!')
try {
const res = await fetch('https://api.example.com/data')
data.value = await res.json()
} finally {
loading.value = false
}
})

onUpdated(() => {
// Dipanggil setiap kali data berubah & DOM diupdate
```

```
console.log('DOM sudah diperbarui')
})

onUnmounted(() => {
// Bersihkan resources: timer, event listener, dll
console.log('Komponen dihapus, bersihkan resources!')
})
</script>
```

6.2 Composables — Logika yang Dapat Digunakan Ulang

Composables adalah fungsi yang menggunakan Composition API untuk mengekstrak logika yang dapat digunakan ulang di berbagai komponen.

useCounter.js

```
// src/composables/useCounter.js
import { ref, computed } from 'vue'

export function useCounter(nilaiAwal = 0) {
const hitungan = ref(nilaiAwal)

const ganda = computed(() => hitungan.value * 2)

function tambah(jumlah = 1) { hitungan.value += jumlah }
function kurang(jumlah = 1) { hitungan.value -= jumlah }
function reset() { hitungan.value = nilaiAwal }

return { hitungan, ganda, tambah, kurang, reset }
}
```

KomponenApapun.vue

```
// Penggunaan di komponen manapun
<script setup>
import { useCounter } from '@/composables/useCounter'

// Setiap komponen mendapat state-nya sendiri
const { hitungan, ganda, tambah, kurang, reset } = useCounter(10)
</script>

<template>
<p>Hitungan: {{ hitungan }} | Ganda: {{ ganda }}</p>
<button @click="tambah()">+1</button>
<button @click="tambah(5)">+5</button>
<button @click="kurang()">-1</button>
<button @click="reset">Reset</button>
</template>
```

Bab 7 — Direktif Lanjutan

7.1 v-model — Two-Way Data Binding

v-model membuat binding dua arah antara data dan elemen form. Perubahan di UI langsung tercermin di data, begitu pula sebaliknya.

VModel.vue

```
<template>
<div>
<!-- Text input -->
<input v-model="nama" placeholder="Nama kamu">
<p>Halo, {{ nama }}!</p>

<!-- Textarea -->
<textarea v-model="pesan"></textarea>

<!-- Checkbox -->
<input type="checkbox" v-model="setuju"> Saya setuju

<!-- Multiple checkbox ke array -->
<input type="checkbox" value="vue" v-model="dipilih"> Vue
<input type="checkbox" value="react" v-model="dipilih"> React
<p>Dipilih: {{ dipilih }}</p>

<!-- Select / dropdown -->
<select v-model="kota">
<option value="sby">Surabaya</option>
<option value="jkt">Jakarta</option>
<option value="bdg">Bandung</option>
</select>

<!-- v-model Modifiers -->
<input v-model.trim="judul"> <!-- trim whitespace -->
<input v-model.number="umur"> <!-- konversi ke number -->
<input v-model.lazy="catatan"> <!-- update saat blur -->
</div>
</template>

<script setup>
import { ref } from 'vue'

const nama = ref('')
const pesan = ref('')
```

```
const setuju = ref(false)
const dipilih = ref([])
const kota = ref('sby')
const judul = ref('')
const umur = ref(0)
const catatan = ref('')
</script>
```

7.2 v-for — Render Daftar

Gunakan **v-for** untuk merender daftar item. Selalu tambahkan **:key** untuk performa optimal.

VFor.vue

```
<template>
<div>
<!-- Array sederhana -->
<ul>
<li v-for="buah in daftarBuah" :key="buah">{{ buah }}</li>
</ul>

<!-- Array of objects -->
<div v-for="produk in produkList" :key="produk.id">
<h3>{{ produk.nama }}</h3>
<p>Rp {{ produk.harga.toLocaleString('id-ID') }}</p>
</div>

<!-- Dengan index -->
<div v-for="(item, index) in daftarBuah" :key="index">
{{ index + 1 }}. {{ item }}
</div>

<!-- Iterasi objek -->
<div v-for="(nilai, kunci) in biodata" :key="kunci">
{{ kunci }}: {{ nilai }}
</div>

<!-- v-for dengan v-if (gunakan computed lebih baik) -->
<div v-for="p in produkList" :key="p.id" v-if="p.tersedia">
{{ p.nama }} — Tersedia
</div>
</div>
</template>

<script setup>
```

```
import { ref } from 'vue'

const daftarBuah = ref(['Mangga', 'Jeruk', 'Pisang', 'Apel'])
const produkList = ref([
{ id: 1, nama: 'Laptop', harga: 8500000, tersedia: true },
{ id: 2, nama: 'Mouse', harga: 150000, tersedia: false },
{ id: 3, nama: 'Keyboard', harga: 350000, tersedia: true },
])
const biodata = ref({ nama: 'Budi', kota: 'Surabaya', umur: 25 })
</script>
```

7.3 v-if, v-else-if, v-else, v-show

Kondisional.vue

```
<template>
<div>
<!-- v-if: elemen dihapus dari DOM jika false -->
<div v-if="status === 'login'">
<h2>Selamat Datang, {{ nama }}!</h2>
</div>
<div v-else-if="status === 'loading'">
<p>Memuat data...</p>
</div>
<div v-else>
<p>Silakan login terlebih dahulu.</p>
</div>

<!-- v-show: elemen tetap ada di DOM, hanya di-hide (display:none) -->
<!-- Lebih efisien jika toggle sering -->
<div v-show="modalTerbuka">
<p>Ini adalah modal dialog</p>
<button @click="modalTerbuka = false">Tutup</button>
</div>
</div>
</template>

<script setup>
import { ref } from 'vue'

const status = ref('logout')
const nama = ref('Budi')
const modalTerbuka = ref(false)
</script>
```

■ **TIP: v-if vs v-show:**

• **v-if**: elemen dihapus/dibuat ulang di DOM. Cocok jika kondisi jarang berubah.

• **v-show**: hanya mengubah CSS display. Cocok untuk toggle yang sering.

Bab 8 — Vue Router

Vue Router adalah library resmi untuk routing di Vue.js. Memungkinkan pembuatan Single Page Application (SPA) dengan navigasi multi-halaman.

8.1 Instalasi & Konfigurasi

Terminal

```
# Install Vue Router
npm install vue-router@4
```

src/router/index.js

```
// src/router/index.js
import { createRouter, createWebHistory } from 'vue-router'
import BerandaView from '@/views/BerandaView.vue'

const router = createRouter({
history: createWebHistory(),
routes: [
{
path: '/',
name: 'beranda',
component: BerandaView
},
{
path: '/tentang',
name: 'tentang',
// Lazy loading: komponen dimuat hanya saat dibutuhkan
component: () => import('@/views/TentangView.vue')
},
{
// Route parameter dinamis
path: '/produk/:id',
name: 'detail-produk',
component: () => import('@/views/DetailProduk.vue')
},
{
// Redirect
path: '/home',
redirect: '/'
```

```
  },
  {
  // Halaman 404
  path: '/:pathMatch(.*)*',
  name: 'not-found',
  component: () => import('@/views/NotFound.vue')
  }
  ]
  })
  export default router
```

src/main.js

```
// src/main.js — Daftarkan router ke aplikasi
import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

const app = createApp(App)
app.use(router)
app.mount('#app')
```

8.2 Navigasi di Template

App.vue

```
<template>
<nav>
<!-- RouterLink menggantikan tag <a> -->
<RouterLink to="/">Beranda</RouterLink>
<RouterLink :to="{ name: 'tentang' }">Tentang</RouterLink>
<RouterLink :to="{ name: 'detail-produk', params: { id: 42 } }">
Produk #42
</RouterLink>

<!-- active-class otomatis ditambahkan ke link aktif -->
</nav>

<!-- RouterView: tempat komponen halaman dirender -->
<RouterView />
</template>
```

8.3 Navigasi Programatik & Route Params

DetailProduk.vue

```
<script setup>
import { useRouter, useRoute } from 'vue-router'

const router = useRouter() // untuk navigasi
const route = useRoute() // untuk baca info route

// Baca parameter URL: /produk/42 → route.params.id = "42"
console.log('ID Produk:', route.params.id)

// Baca query string: /cari?q=laptop → route.query.q = "laptop"
console.log('Query:', route.query.q)

// Navigasi programatik
function keProduk(id) {
router.push({ name: 'detail-produk', params: { id } })
}

function kembali() {
router.back()
}

function keBerandaReplace() {
// replace: tidak menambah history baru
router.replace('/')
}
</script>
```

8.4 Navigation Guards

Guard digunakan untuk proteksi halaman, misalnya memastikan pengguna sudah login.

```
router/index.js
// src/router/index.js — Global guard
router.beforeEach((to, from, next) => {
const sudahLogin = localStorage.getItem('token')

if (to.meta.requiresAuth && !sudahLogin) {
// Belum login, redirect ke halaman login
next({ name: 'login' })
} else {
next() // Lanjutkan navigasi
}
})

// Route dengan meta data
const routes = [
{
```

```
  path: '/dashboard',
  component: DashboardView,
  meta: { requiresAuth: true } // Route ini butuh login
  }
  ]
```

Bab 9 — Pinia (State Management)

Pinia adalah state management resmi untuk Vue 3. Digunakan untuk berbagi data (state) antar komponen yang tidak memiliki hubungan parent-child langsung.

9.1 Instalasi & Setup

Terminal

```
npm install pinia
```

main.js

```
// src/main.js
import { createApp } from 'vue'
import { createPinia } from 'pinia'
import App from './App.vue'

const app = createApp(App)
app.use(createPinia())
app.mount('#app')
```

9.2 Membuat Store

src/stores/pengguna.js

```
// src/stores/pengguna.js
import { defineStore } from 'pinia'
import { ref, computed } from 'vue'

// Menggunakan Composition API style (direkomendasikan)
export const usePenggunaStore = defineStore('pengguna', () => {
// State
const nama = ref('')
const email = ref('')
const token = ref(null)

// Getters (computed)
const sudahLogin = computed(() => !!token.value)
const inisial = computed(() =>
nama.value ? nama.value[0].toUpperCase() : '?'
)

// Actions
async function login(emailInput, password) {
try {
```

```
// Simulasi API call
const res = await fetch('/api/login', {
method: 'POST',
body: JSON.stringify({ email: emailInput, password })
})
const data = await res.json()
nama.value = data.nama
email.value = data.email
token.value = data.token
} catch (err) {
console.error('Login gagal:', err)
}
}

function logout() {
nama.value = ''
email.value = ''
token.value = null
}

return { nama, email, token, sudahLogin, inisial, login, logout }
})
```

9.3 Menggunakan Store di Komponen

NavBar.vue

```
<script setup>
import { usePenggunaStore } from '@/stores/pengguna'

const penggunaStore = usePenggunaStore()

// Akses state langsung
console.log(penggunaStore.nama)
console.log(penggunaStore.sudahLogin)

// Panggil action
async function handleLogin() {
await penggunaStore.login('budi@email.com', 'password123')
}
</script>

<template>
<div>
<div v-if="penggunaStore.sudahLogin">
```

```
<p>Halo, {{ penggunaStore.nama }}! ({{ penggunaStore.inisial }})</p>
<button @click="penggunaStore.logout()">Logout</button>
</div>
<div v-else>
<button @click="handleLogin">Login</button>
</div>
</div>
</template>
```

Bab 10 — Tips & Best Practices

10.1 Konvensi Penamaan

Item	Konvensi	Contoh
Nama file komponen	PascalCase	KartuPengguna.vue
Nama komponen di template	PascalCase	<KartuPengguna />
Nama props	camelCase (JS) / kebab-case (HTML)	namaLengkap / nama-lengkap
Nama event	camelCase	dataKirim, formSubmit
Nama store file	camelCase	usePengguna.js
Nama composable	use + PascalCase	useCounter.js
Nama variabel reaktif	camelCase	daftarProduk, isLoading

10.2 Tips Performa

■ **Gunakan :key yang unik**	Selalu gunakan :key pada v-for dengan ID unik (bukan index) untuk mencegah re-render yang tidak perlu.
■ **Lazy Loading Komponen**	Import komponen besar secara lazy: defineAsyncComponent(() => import('./Besar.vue'))
■ **v-once untuk konten statis**	Tambahkan v-once pada elemen yang tidak pernah berubah agar hanya direnderer sekali.
■ **Hindari v-if + v-for bersama**	Jangan kombinasikan v-if dan v-for di elemen yang sama. Gunakan computed atau pisahkan template.
■ **Gunakan shallowRef/shallowReactive**	Untuk objek besar yang tidak butuh deep reactivity, gunakan shallowRef() untuk performa lebih baik.

10.3 Tools & Ekosistem Vue yang Wajib Diketahui

Tool	Fungsi	Link
Vite	Build tool super cepat	`vitejs.dev`
Vue DevTools	Debug Vue di browser (ekstensi)	`devtools.vuejs.org`

Pinia	State management resmi Vue 3	`pinia.vuejs.org`
Vue Router	Routing untuk SPA	`router.vuejs.org`
VueUse	Kumpulan 200+ composables siap pakai	`vueuse.org`
Nuxt.js	Framework full-stack berbasis Vue	`nuxt.com`
Vitest	Unit testing untuk Vue & Vite	`vitest.dev`
Quasar	UI framework komponen lengkap	`quasar.dev`
Naive UI	Komponen UI modern untuk Vue 3	`naiveui.com`

■ **TIP:** Mulai belajar Vue dari dokumentasi resminya di **vuejs.org** — dokumentasinya tersedia dalam banyak bahasa dan sangat mudah dipahami, bahkan untuk pemula sekalipun!

Selamat Belajar!

Kamu telah menyelesaikan rangkuman Vue.js ini. Sekarang saatnya praktik!

1■■ Buat proyek baru dengan **npm create vue@latest**

2■■ Buat komponen sederhana dan coba semua direktif

3■■ Tambahkan Vue Router untuk navigasi multi-halaman

4■■ Gunakan Pinia untuk mengelola state aplikasi

5■■ Deploy proyekmu ke Vercel atau Netlify

Sumber Belajar Lanjutan

■ Dokumentasi Resmi: vuejs.org/guide

■ Tutorial Interaktif: vuejs.org/tutorial

■ Vue School: vueschool.io

■ Forum Komunitas: forum.vuejs.org

■ GitHub Vue.js: github.com/vuejs/core