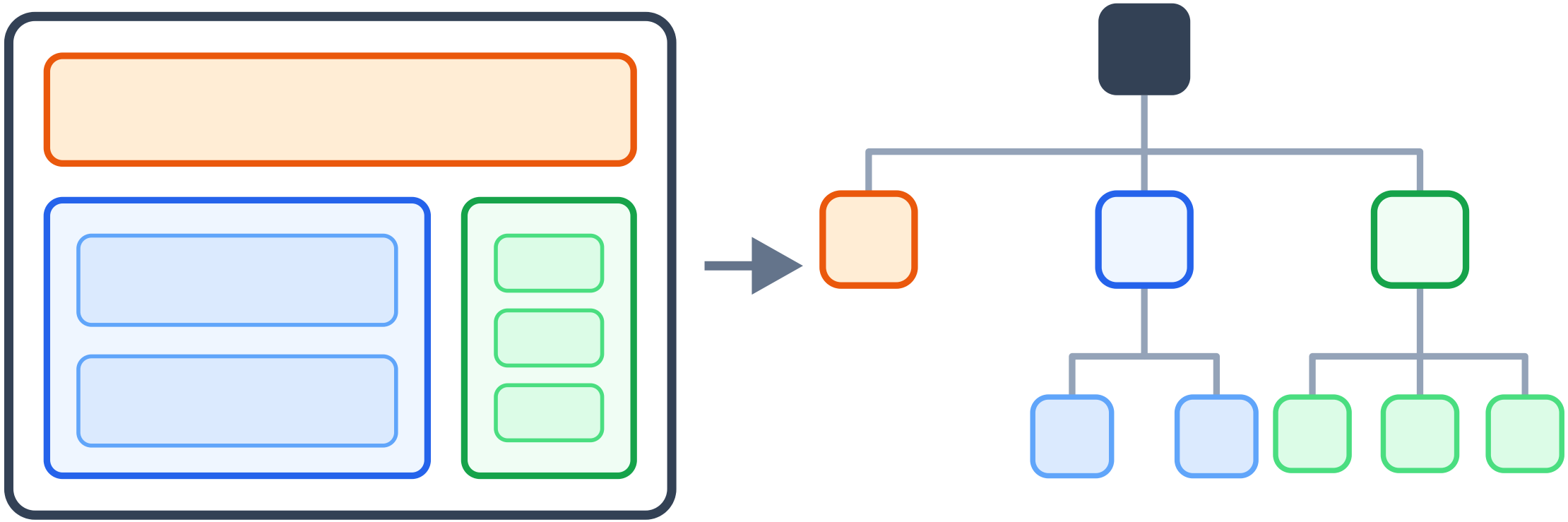
Сучасні каркаси web-додатків

Vue 3: компоненти, шаблони та реактивність

Від створення застосунку до інтерактивного інтерфейсу

Лекція 2 · Лісняк Андрій Олександрович

Інтерфейс стає деревом компонентів



Великі області сторінки стають компонентами, а повторювані елементи — їхніми дочірніми екземплярами.

Після лекції ви зберете інтерактивний компонент

Запустити

Створити проєкт, знайти точку входу та змонтувати кореневий компонент.

Описати

Прочитати різні форми компонента й написати SFC через script setup.

Зв’язати

Передати дані до шаблону інтерполяцією, атрибутами й директивами.

Змінити

Оголосити реактивний стан і похідні значення без ручного керування DOM.

Обробити

Реагувати на DOM-події та повідомлення дочірніх компонентів.

Очистити

Підключити зовнішній ресурс під час монтування й прибрати його при демонтажі.

create-vue готує проєкт, Vite запускає середовище

1. **Створити** Виконати npm create vue@latest і вибрати мінімальну конфігурацію.
2. **Встановити** Перейти в каталог проєкту та виконати npm install.
3. **Запустити** npm run dev піднімає локальний сервер і показує адресу.
4. **Перевірити** Змінити App.vue та побачити оновлення без ручного перезавантаження.

Джерело: Vue.js · Швидкий старт — https://ua.vuejs.org/guide/quick-start.html

mount визначає місце застосунку в DOM



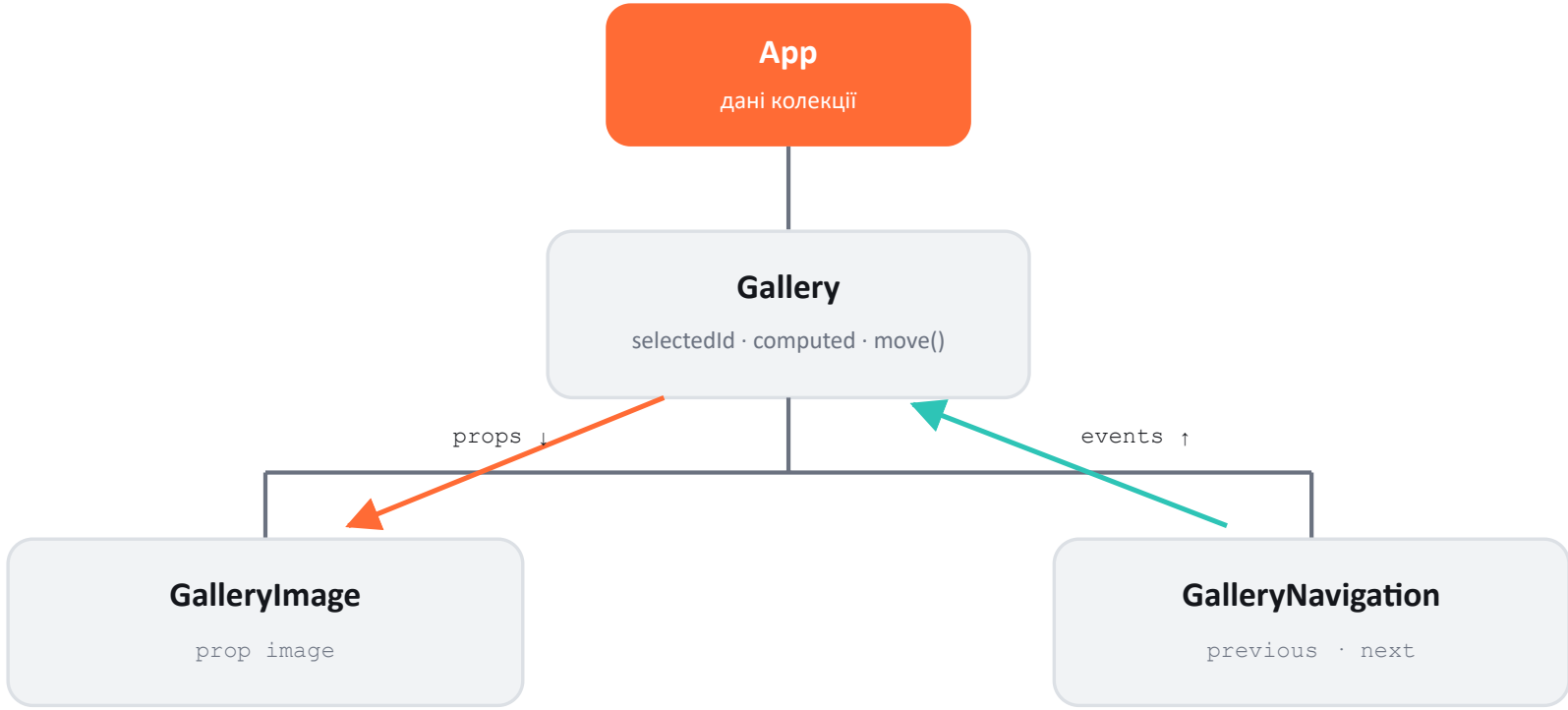
До виклику mount застосунок створено, але його інтерфейс ще не відтворено.

До виклику mount застосунок створено, але його інтерфейс ще не відтворено.

Кожен стартовий файл має одну помітну роль

Файл	Відповідальність
index.html	Містить DOM-контейнер і підключає модуль src/main.js.
src/main.js	Створює екземпляр Vue та монтує кореневий компонент.
src/App.vue	Описує кореневий інтерфейс і підключає дочірні компоненти.
src/components/	Зберігає повторно використовувані частини інтерфейсу.
package.json	Фіксує залежності й команди dev та build.
vite.config.js	Налаштовує Vite та плагін обробки файлів .vue.

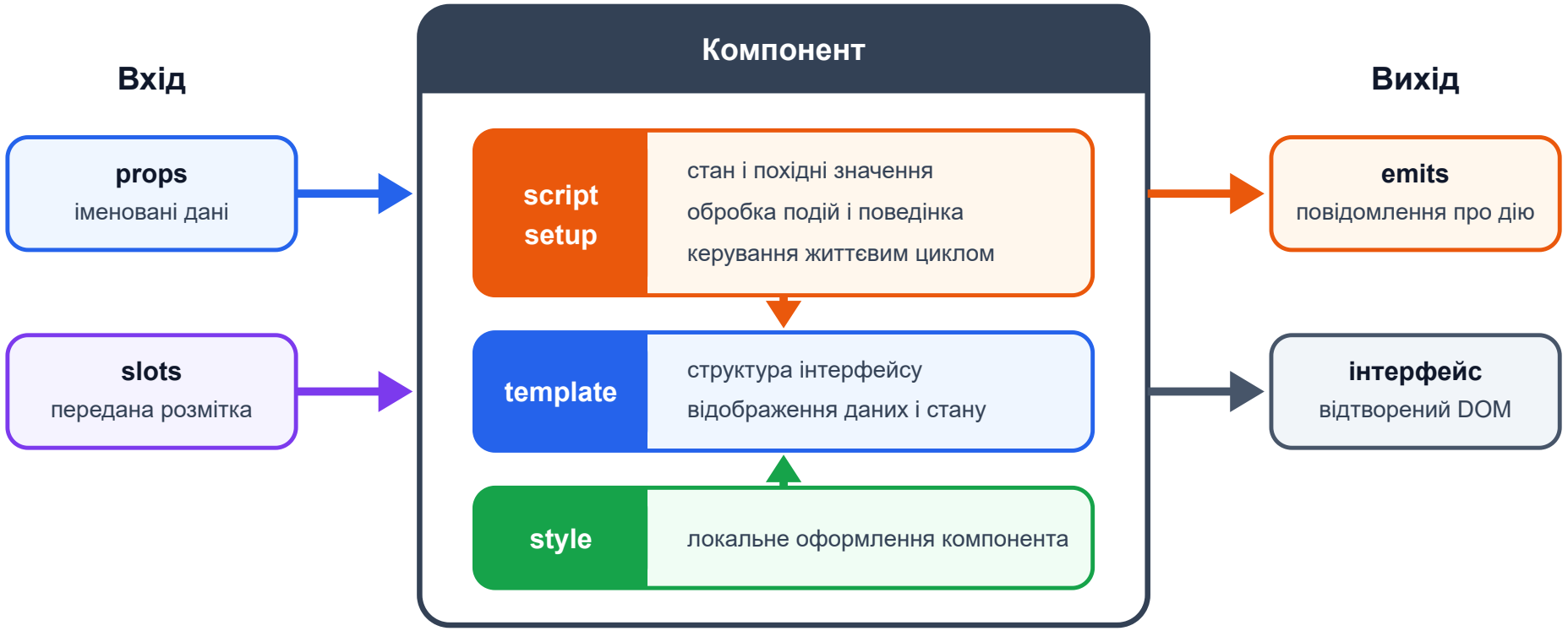
Дерево компонентів має явного власника стану



Власник стану передає дані вниз; дочірній компонент повідомляє про намір угору.

Дані рухаються до дочірніх компонентів, а повідомлення про дії — до власника стану.

SFC тримає структуру, логіку й оформлення поруч



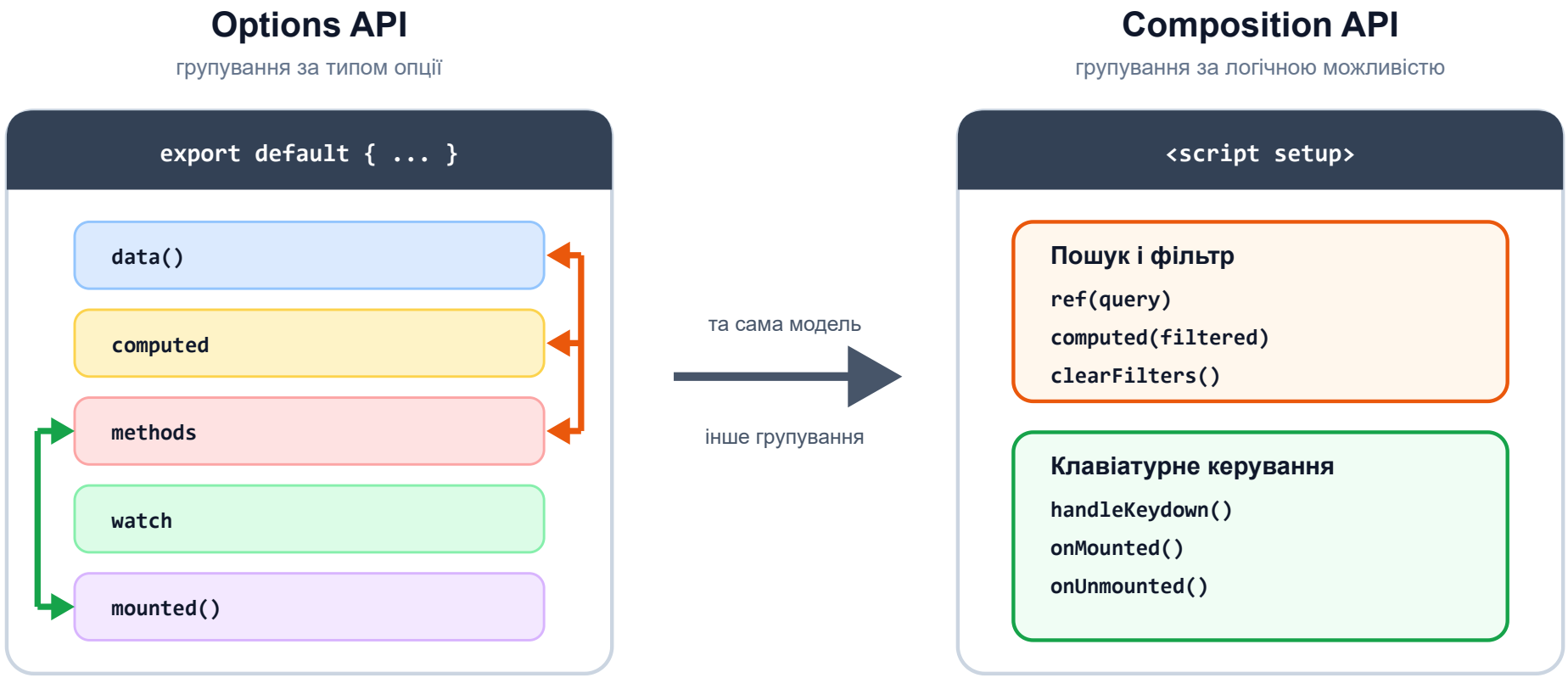
Компонент перетворює вхідні дані та вміст на інтерфейс і повідомлення про дії.

Props і emit утворюють контракт компонента



Батьківський компонент володіє станом, а дочірній дотримується оголошеного контракту.

API по-різному групують ту саму логіку



> Обидва API спираються на ту саму компонентну й реактивну модель Vue.

Options API і setup організують логіку по-різному

Options API · src/Options.vue

```
<script>
export default {
  data() {
    return { count: 0 }
  },
  methods: {
    increment() {
      this.count++
    },
  },
}
</script>

<template>
```

setup() · src/Setup.vue

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

export default {
  setup() {
    const count = ref(0)

    function increment() {
      count.value++
    }

    return { count, increment }
  },
}
```

```
npm install && npm run dev
```

script setup прибирає службову оболонку

script setup · src/ScriptSetup.vue

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

const count = ref(0)

function increment() {
  count.value++
}
</script>

<template>
  <button type="button" @click="increment">script setup: {{ count }}</but
</template>
```

JS-об’єкт · src/Object.js

```
import { ref } from 'vue'

export default {
  setup() {
    const count = ref(0)
    const increment = () => count.value++

    return { count, increment }
  },
  template: `
    <button type="button" @click="increment">
      JS-об’єкт: {{ count }}
    </button>
  `,
```

```
npm install && npm run dev
```

У курсі пишемо Composition API через script setup

Читаємо обидва

Options API залишається чинним і часто трапляється в наявних проєктах.

Пишемо одним стилем

Нові навчальні приклади використовують Composition API та script setup.

Не змішуємо без причини

Один компонент має одну зрозумілу систему організації логіки.

Точка входу створює і монтує застосунок

src/main.js

```
import { createApp } from 'vue'

import App from './App.vue'

import './styles/main.scss'



createApp(App).mount('#app')
```

```
npm install && npm run dev
```

Шаблон поєднує HTML із чотирма видами зв'язку

`{{ value }}`	Інтерполює текст і автоматично оновлюється разом зі значенням.
`:attribute`	Передає JavaScript-значення в атрибут, властивість або prop компонента.
`v-directive`	Застосовує спеціальну реактивну поведінку до елемента чи компонента.
`@event`	Запускає вираз або функцію у відповідь на DOM-подію чи подію компонента.

> У шаблоні дозволені вирази, але не довільні послідовності інструкцій.

Один шаблон показує v-model, подію, умову і список

Поля і подія · src/Filter.vue · рядки 35–46

```
<template>
  <section>
    <label>Пошук <input v-model.trim="query"></label>
    <label>
      Тема
      <select v-model="topic">
        <option value="all">Усі</option>
        <option value="vue">Vue</option>
        <option value="html">HTML</option>
      </select>
    </label>
```

Умова і список · src/Filter.vue · рядки 48–54

```
    <p>Знайдено: {{ filteredMaterials.length }} · {{ totalM
    <ul v-if="filteredMaterials.length">
      <li v-for="material in filteredMaterials" :key="mater
        {{ material.title }} — {{ material.minutes }} хв
      </li>
    </ul>
    <p v-else>За заданими умовами матеріалів немає.</p>
```

```
npm install && npm run dev
```

Кожна директива відповідає на окреме питання

v-bind

Яке динамічне значення отримує атрибут або prop?

v-on

Що виконати, коли відбулася подія?

v-model

З яким станом синхронізувати поле форми?

v-if

Чи має гілка існувати в DOM?

v-show

Чи має готовий елемент бути видимим?

v-for

Для яких елементів колекції повторити шаблон?

Реактивний стан пов’язує дані, обчислення й DOM



Vue відстежує читання стану та оновлює залежний код після його зміни.

ref зберігає вибір, computed його виводить

src/components/Gallery.vue · рядки 16–27

```
// Стан галереї — вибір користувача, і тільки він.
const selectedId = ref(null)


// Решта обчислюється. Похідне значення не зберігають у стані:
// збережена копія рано чи пізно розійдеться з оригіналом.
const selectedIndex = computed(() => {
  const index = props.images.findIndex((image) => image.id === selectedId.value)
  return index < 0 ? 0 : index
})
const selectedImage = computed(() => props.images[selectedIndex.value] ?? null)
const total = computed(() => props.images.length)
```

```
npm install && npm run dev
```

ref і reactive відрізняються формою доступу

ref()	reactive()
Приймає значення будь-якого типу	Приймає лише об’єктні типи
У JavaScript доступ через .value	Властивості читаються без .value
Значення можна повністю замінити	Реактивність пов’язана з proxy-об’єктом
Добрий типовий вибір для стану	Зручний для пов’язаної групи полів
У шаблоні верхній рівень розгортається	Доступ у шаблоні як до звичайного об’єкта

> Для базових прикладів курсу починаємо з ref; reactive вводимо там, де стан природно є об’єктом.

computed зберігає похідне значення

Зберігаємо джерело істини; кількість, фільтр і форматований підпис обчислюємо.

Залежності

Getter читає реактивні значення, які Vue відстежує автоматично.

Кеш

Повторне читання повертає попередній результат, доки залежності не змінилися.

Чистота

Getter обчислює результат, але не змінює стан і не запускає зовнішні дії.

Фільтр і підсумок залежать від стану

Джерельний стан · src/Filter.vue · рядки 10–11

```
const query = ref('')

const topic = ref('all')
```

Похідний стан · src/Filter.vue · рядки 13–20

```
const filteredMaterials = computed(() => materials.filter((material) => {
  const matchesQuery = material.title.toLowerCase().includes(query.value.
  const matchesTopic = topic.value === 'all' || material.topic === topic.
  return matchesQuery && matchesTopic
}))


const totalMinutes = computed(() => filteredMaterials.value
  .reduce((sum, material) => sum + material.minutes, 0))
```

```
npm install && npm run dev
```

DOM і компонент надсилають різні події

DOM-подія	Подія компонента	
Надходить від button	input або іншого DOM-вузла	Надсилається дочірнім Vue-компонентом
Слухаємо як @click або @input	Слухаємо як @select або @next	
Обробник може отримати нативний Event	Payload визначає контракт компонента	
Модифікатори .prevent .stop .once відділяють деталі DOM	defineEmits документує дозволені події	

Дитина надсилає подію власнику стану

src/components/GalleryNavigation.vue · рядки 7–26

```
defineProps({
  disabled: { type: Boolean, default: false },
})

// Перелік подій оголошено явно: це такий самий контракт компонента,
// як і props, і його видно з першого рядка.
const emit = defineEmits(['previous', 'next'])
</script>

<template>
  <div class="gallery-nav">
```

```
npm install && npm run dev
```

v-model синхронізує поле зі станом

Явний запис	Скорочений запис
:value="query"	v-model="query"
@input="query = $event.target.value"	Значення оновлюється відповідною подією автоматично
Видно окремий напрям даних і окрему подію	Видно намір двостороннього зв’язування
Потрібно для нетипової поведінки	Зручний типовий вибір для елементів форми

> На цьому занятті v-model застосовуємо до нативних полів; компонентний v-model розглядається окремо.

v-model враховує тип поля

text · textarea

Працюють із рядком через value та подію input.

checkbox · radio

Використовують checked; група checkbox може формувати масив.

select

Зберігає value вибраного option; multiple повертає масив.

.trim

Прибирає пробіли на початку й наприкінці введеного рядка.

.number

Намагається перетворити введене значення на число.

.lazy

Синхронізує стан за change, а не після кожної події input.

Хуки підключають і очищують ресурси



Зовнішній listener, timer або subscription підключають під час монтування і прибирають під час демонтажу.

> Зовнішній ресурс має завершити життя разом із компонентом, який його створив.

Зовнішній listener потребує очищення

Filter · src/Filter.vue · рядки 27–32

```
function handleKeydown(event) {
  if (event.key === 'Escape') clearFilters()
}

onMounted(() => window.addEventListener('keydown', handleKeydown))
onUnmounted(() => window.removeEventListener('keydown', handleKeydown))
```

Перевірка демонтажу · src/App.vue · рядки 24–30

```
    <section>
      <h2>Реактивний фільтр</h2>
      <button type="button" @click="showFilter = !showFilter">
        {{ showFilter ? 'Сховати' : 'Показати' }} фільтр
      </button>
      <FilterDemo v-if="showFilter" />
    </section>
```

```
npm install && npm run dev
```

Gallery зберігає лише вибір користувача

src/components/Gallery.vue · рядки 16–34

```
// Стан галереї — вибір користувача, і тільки він.
const selectedId = ref(null)


// Решта обчислюється. Похідне значення не зберігають у стані:
// збережена копія рано чи пізно розійдеться з оригіналом.
const selectedIndex = computed(() => {
  const index = props.images.findIndex((image) => image.id === selectedId.value)
  return index < 0 ? 0 : index
})
const selectedImage = computed(() => props.images[selectedIndex.value] ?? null)
const total = computed(() => props.images.length)
const isSingle = computed(() => total.value < 2)


// Залишок від ділення робить перегляд кільцевим: після останнього кадру
```

```
npm install && npm run dev
```

Props і події з’єднують компоненти Gallery

src/components/Gallery.vue · рядки 65–77

```
    <GalleryImage :image="selectedImage" />



    <div class="gallery__details">
      <FrameInfo
        :image="selectedImage"
        :position="selectedIndex + 1"
        :total="total"
      />
      <GalleryNavigation
        :disabled="isSingle"
        @previous="move(-1)"
        @next="move(1)"
      />
```

```
npm install && npm run dev
```