ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
Сучасні каркаси web-додатків
Лісняк Андрій Олександрович

Лабораторна робота 2. Компоненти та реактивність

Мета

Навчитися створювати компоненти з використанням механізмів реактивності.

Теоретичні відомості

Компонент Vue — це самостійна повторно використовувана частина інтерфейсу, яка поєднує шаблон, логіку поведінки та, за потреби, локальне оформлення. Компонент приховує деталі реалізації власної поведінки. У застосунку компоненти утворюють дерево: кореневий компонент підключає дочірні, а ті можуть складатися з іще менших частин (рис. 1).

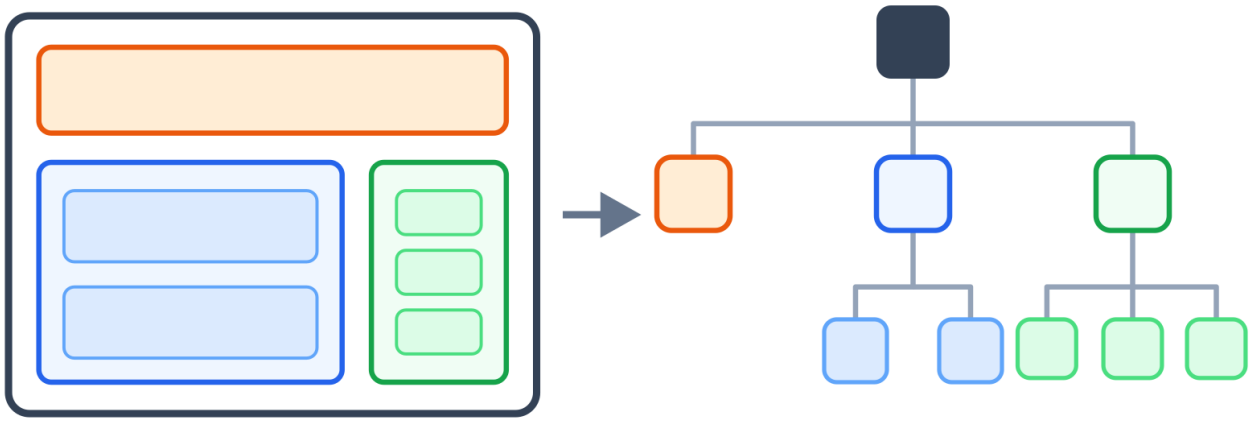

Рисунок 1 — Від компонування інтерфейсу до дерева компонентів

Основні концепції компонентної архітектури:
- складний інтерфейс збирається з невеликих компонентів із чіткими відповідальностями;
- один опис компонента можна використати багато разів і кожне використання створює окремий екземпляр;
- локальний реактивний стан належить конкретному екземпляру;
- вхідні дані передаються через пропси, довільний вміст через слоти, а вихідні повідомлення — через події компонента;
- батьківський компонент передає дані вниз, а дочірній повідомляє про намір користувача вгору;
- кожен екземпляр створюється, монтується, оновлюється та демонтується.

Однофайловий компонент, або **SFC**, має розширення `.vue` та блоки `<script setup>`, `<template>` і, за потреби, `<style scoped>`. Шаблон описує

представлення, script — стан і поведінку, а scoped-стилі обмежують CSS цим компонентом. Верхньорівневі оголошення з `<script setup>` доступні шаблону автоматично (рис. 2).



Рисунок 2 — Компонент як перетворення вхідного контракту на інтерфейс і події

Vue 3 підтримує два способи організації логіки компонента. В **Options API** оголошення розкладають за типами опцій:

- початковий стан міститься в `data`,
- похідні значення — у `computed`,
- функції — у `methods`,
- спостереження — у `watch`,
- код монтування — у `mounted`.

У **Composition API** пов'язані стан, обчислення, функції та lifecycle-хуки можна розмістити поруч усередині `setup()` або `<script setup>`. Це спрощує читання логіки окремої можливості та дає змогу за потреби винести її в композиційну функцію. Обидва API спираються на ту саму компонентну модель і не є різними версіями Vue. У роботах курсу, як правило, використовується Composition API з `<script setup>`, але код Options API потрібно вміти читати.

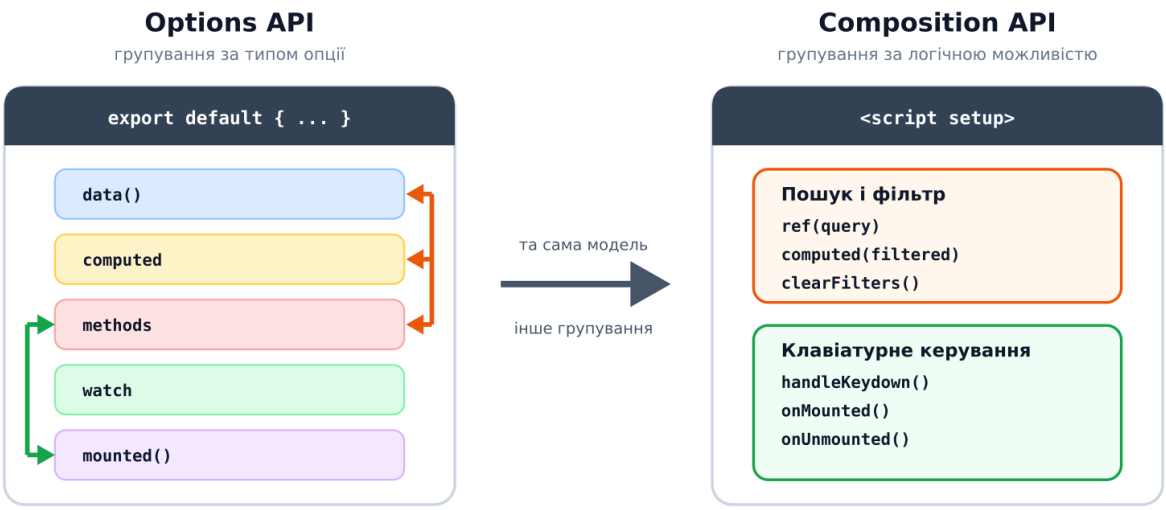

Рисунок 3 — Групування логіки за типом опції та за логічною можливістю

Вхідні й вихідні параметри

Основним вхідним каналом є **пропси** — іменовані значення, які батько передає дочірньому компоненту. Пропс має назву, очікуваний тип і може мати ознаку обов'язковості, значення за замовчуванням або валідатор. Дочірній компонент читає пропси, але **не змінює їх**. **Слоти** — другий вхідний канал, який передає не значення, а фрагмент розмітки. Вихідним каналом є **подія компонента**. Подія має назву й може переносити payload — дані про виконану дію. Це не значення, яке функція повертає батьківському компоненту, і не DOM-подія, що автоматично спливає крізь усе дерево. Батько має явно підписатися на неї та самостійно змінити свій стан.



Рисунок 4 — Вхідні пропси та вихідна подія дочірнього компонента

Оголошення контракту у Vue

У Vue 3 контракт можна задати опціями `props` та `emits` або макросами `defineProps` і `defineEmits`. При використанні Composition API з `<script`

`setup>` макроси не потрібно імпортувати оскільки компілятор Vue обробляє їх під час складання SFC.

Дочірній компонент оголошує вхідні пропси і допустиму вихідні події:

```
<script setup>
// Дочірній компонент - MaterialCard
defineProps({
  title: { type: String, required: true },
  topic: { type: String, required: true },
  minutes: { type: Number, required: true },
})

const emit = defineEmits(['select'])
</script>

<template>
  <article>
    <h3>{{ title }}</h3>
    <p>{{ topic }} · {{ minutes }} хв</p>
    <button type="button" @click="emit('select',
title)">Обрати</button>
  </article>
</template>
```

Батьківський компонент імпортує SFC, передає динамічні значення скороченням `v-bind` (`:`) і слухає подію скороченням `v-on` (`@`):

```
<script setup>
// Батьківський компонент
import { ref } from 'vue'
import MaterialCard from './MaterialCard.vue'

const selectedTitle = ref('')
</script>

<template>
  <MaterialCard
    :title="material.title"
    :topic="material.topic"
    :minutes="material.minutes"
    @select="selectedTitle = $event"
  />
</template>
```

У шаблоні імпортований `MaterialCard` стає тегом компонента. Вираз після двокрапки обчислюється як JavaScript і передає результат у пропс. Значення, яке картка додала до `emit`, доступне обробнику як `$event`. Отже, картка повідомляє про вибір, а каталог лишається єдиним власником `selectedTitle`.

Реактивність

У Composition API реактивність стану реалізується двома основними способами: за допомогою `ref()` та `reactive()`.

`ref()` створює реактивне посилання на значення будь-якого типу. У JavaScript значення читають і змінюють через властивість `.value`. У шаблоні Vue автоматично розгортає верхньорівневе реактивне посилання, тому звертатися до `.value` не потрібно.

`reactive()` створює реактивний об'єкт на основі JavaScript Proxy. Цей спосіб використовують для об'єктів, масивів та інших колекцій. Реактивність є глибокою: Vue відстежує зміни не лише властивостей самого об'єкта, а й вкладених об'єктів. Для роботи зі станом потрібно використовувати Proxy, повернений функцією `reactive()`.



Рисунок 5 — Від оголошення реактивного стану до оновлення інтерфейсу

Двостороннє зв’язування

Двостороннє зв’язування синхронізує реактивний стан і значення елемента форми у двох напрямках: зміна стану оновлює поле, а введення користувача оновлює стан. У Vue для цього використовують директиву `v-model`. Реактивне значення при цьому лишається єдиним джерелом істини, а директива лише скорочує запис передавання значення та оброблення події.

```
<input v-model.trim="filterForm.query">
<select v-model="filterForm.topic">
  <!-- варіанти тем -->
</select>
```

Для текстового поля `v-model` передає поточний стан у властивість `value` і слухає подію `input`. Для елемента `select` значення так само передається через `value`, але зміна зчитується під час події `change`. Той самий обмін можна записати без директиви:

```
<input
```

```
  :value="filterForm.query"
  @input="filterForm.query = $event.target.value.trim()"
>

<select
  :value="filterForm.topic"
  @change="filterForm.topic = $event.target.value"
>
  <!-- варіанти тем -->
</select>
```

У першому прикладі модифікатор `.trim` прибирає пробіли на початку та в кінці рядка перед записом до стану. Без `v-model` це перетворення потрібно виконати явно в обробнику `input`. Такий розгорнутий запис корисний, коли перед оновленням стану потрібно перевірити або перетворити введене значення.

Для власного компонента `v-model` використовує пропс `modelValue` та подію `update:modelValue`. Тому запис `<FilterField v-model="query" />` можна реалізувати без директиви так:

```
<FilterField
  :model-value="query"
  @update:model-value="query = $event"
/>
```

Дочірній компонент не змінює пропс безпосередньо: він показує `modelValue` і надсилає нове значення подією `update:modelValue`. Власником стану лишається батьківський компонент.

Двостороннє зв’язування поля не означає, що фільтр потрібно застосовувати після кожного введеного символу. У цій роботі `filterForm` зберігає поточні значення полів, а подія `apply` передає їх каталогу лише після надсилання форми. Це розділяє редагування форми та застосований стан фільтра.

Функціональні вимоги

- Каталог показує перелік навчальних матеріалів за допомогою окремого компонента MaterialCard
- Вигляд застосунку після перенесення розмітки відповідає статичній заготовці, а персональні дані та середовище в шапці заповнені студентом
- Надсилання форми застосовує пошук за назвою та фільтр за темою без перезавантаження сторінки
- Форма фільтрації реалізована окремим компонентом MaterialFilterForm
- Поруч зі списком показано кількість знайдених матеріалів і їхню сумарну тривалість
- Натискання кнопки картки передає її назву батьківському компоненту й оновлює напис про вибраний матеріал

- Після запуску вибрано перший матеріал Компоненти Vue
- Вибрана картка має окреме візуальне оформлення та напис Обрано на кнопці
- Скидання форми та клавіша Escape очищують введені й застосовані фільтри, а для порожнього результату показано окреме повідомлення

Технічні вимоги

- Компоненти написані як Vue SFC із Composition API та script setup
- MaterialCard оголошує пропси title, topic, description, minutes і selected та подію select і не змінює стан каталогу безпосередньо
- MaterialFilterForm зберігає локальний стан полів, оголошує події apply і clear та не змінює стан каталогу безпосередньо
- Готова розмітка розподілена між App, MaterialCatalog, MaterialFilterForm і MaterialCard за позначеними в заготовці межами, а CSS-класи відповідають BEM
- Поля форми пов'язані зі станом через v-model, submit і reset обробляються з prevent, а список має v-for із унікальним key і умовне відображення через v-if
- Відфільтрований список і сумарна тривалість реалізовані через computed
- Клавіша Escape обробляється на формі та скидає її без перезавантаження
- Tailwind CSS, готові бібліотеки компонентів, HTTP-запити й маршрутизація не використовуються

Необхідне обладнання та матеріали

- Node.js 20 або новіший, npm
- Редактор коду
- Браузер із DevTools
- Статична заготовка інтерфейсу лабораторної роботи

Хід роботи

1. Ознайомлення зі статичною заготовкою

Отримайте заготовку з відкритого репозиторію та скопіюйте її поруч із майбутнім Vue-проєктом:

```
git clone --depth 1 https://github.com/lisnayk/znu-lab-starters.git
../znu-lab-starters
cp -R ../znu-lab-starters/modern-frameworks/labs/02-components-
reactivity/starter ./lab2-layout
```

Відкрийте `lab2-layout/index.html` у браузері. Сторінка має показати шість карток, форму фільтрації, підсумок і кнопки, але не повинна фільтрувати список або змінювати вибраний матеріал. Це очікувано: у заготовці немає Vue та JavaScript-логіки, а `index.html` підключає лише `styles.css`. Окремий файл `materials.js` містить вихідні дані й не виконується статичною сторінкою.

Знайдіть у HTML коментарі, що позначають межі майбутніх `App.vue`, `MaterialCatalog.vue`, `MaterialFilterForm.vue` і `MaterialCard.vue`, а також коментарі `TODO`. Зверніть увагу на BEM-класи: `material-card` — блок, `material-card__title` — елемент, а `material-card--vue` — модифікатор.

2. Створення Vue-проєкту

У корені репозиторію `mjsf-mag` створіть мінімальний Vue-проєкт у каталозі `lab2`:

```
npm create vite@latest lab2 -- --template vue
cd lab2
npm install
```

Скопіюйте готові стилі й дані до проєкту:

```
mkdir -p src/data
cp ../lab2-layout/styles.css src/style.css
cp ../lab2-layout/materials.js src/data/materials.js
```

Переконайтеся, що `src/main.js` імпортує `./style.css`. Перенесіть зовнішню розмітку між коментарями `App.vue` до `src/App.vue` й замініть у шапці плейсхолдери власними ПІБ, групою та відомостями про середовище. Не переносіть `doctype`, `html`, `head` і `body`: вони належать `index.html`, а не SFC.

Запустіть проєкт:

```
npm run dev
```

На цьому етапі важливо зберегти вигляд заготовки. Поведінка з'явиться в наступних етапах.

3. Дані та межі компонентів

Створіть таку структуру в `src`:

```
src/
├── components/
│   ├── MaterialCard.vue
│   ├── MaterialCatalog.vue
│   └── MaterialFilterForm.vue
├── data/
│   └── materials.js
├── App.vue
├── main.js
└── style.css
```

Розподіліть розмітку відповідно до коментарів у статичній заготовці. Перевірте, що скопійований `src/data/materials.js` експортує масив об'єктів із полями `id`, `title`, `topic`, `description` і `minutes`. Контрольний набір наведено у файлі materials.js. Не змінюйте вихідні дані: вони потрібні для однакової перевірки результатів.

Статичні картки з `App.vue` приберіть: після перенесення кожен фрагмент інтерфейсу має існувати лише у своєму компоненті.

4. Компонент картки

У `MaterialCard.vue` оголосіть обов'язкові пропси `title`, `topic`, `description` типу `String` і `minutes` типу `Number`, а також булевий пропс `selected` зі значенням за замовчуванням `false`. Компонент має показувати отримані значення в готовій BEM-розмітці.

Оголосіть подію `select`. Після натискання кнопки `Обрати` передайте з нею назву матеріалу. Картка не повинна безпосередньо змінювати стан каталогу.

Модифікатор теми сформуйте як повний клас:

```
:class="`material-card--${topic}`"
```

Якщо `selected` дорівнює `true`, додайте до картки готовий модифікатор `material-card--selected` і змініть напис кнопки з `Обрати` на `Обрано`. Зв'яжіть атрибут кнопки як `:aria-pressed="selected"`, щоб він передавав обидва стани допоміжним технологіям. Стиль модифікатора вже описаний у `style.css`; власний стан вибору всередині картки створювати не потрібно.

Для цієї роботи перелік тем обмежений контрольованими значеннями з `materials.js`. Не додавайте Tailwind або готову бібліотеку компонентів: оформлення вже є у `style.css`, а завдання полягає в компонентній моделі й поведінці.

5. Компонент форми

Створіть `MaterialFilterForm.vue` та перенесіть до нього розмітку між відповідними коментарями заготовки. Компонент форми не отримує готовий список матеріалів і не фільтрує його безпосередньо.

Створіть через `reactive` об'єкт `filterForm` із полями `query: ''` і `topic: 'all'`. Він зберігає локальний стан елементів форми. Додайте `v-model.trim="filterForm.query"` до поля пошуку та `v-model="filterForm.topic"` до списку тем. Кожне поле має зберегти пов'язаний `label`.

Оголосіть дві події компонента — `apply` і `clear`. Під час надсилання форми функція `submitFilters` передає подією `apply` новий об'єкт зі значеннями `query` і `topic`. Не передавайте назовні сам реактивний Proxy.

Обробіть події самої форми:

```
<form
  class="filters"
  @submit.prevent="submitFilters"
  @reset.prevent="clearFilters"
  @keydown.esc="clearFilters"
>
```

Функція `clearFilters` повертає обидва локальні поля до початкових значень і надсилає подію `clear`. Її викликають `reset` і Escape. Не додавайте окремий обробник `click` до кнопки `Очистити`: кнопка з `type="reset"` уже створює потрібну подію форми. Модифікатор `.prevent` не дозволяє браузеру перезавантажити сторінку після надсилання чи скидання.

6. Стан каталогу та обчислювані властивості

У `MaterialCatalog.vue` імпортуйте дані, `MaterialFilterForm` і `MaterialCard`. Створіть через `reactive` об'єкт `activeFilters` із полями `query: ''` і `topic: 'all'`. Функція `applyFilters` приймає payload події форми й копіює обидва значення до `activeFilters`; `clearFilters` повертає їх до початкових значень.

Назву вибраного матеріалу зберігайте окремо в `selectedTitle`. Початково виберіть перший елемент масиву:

```
const selectedTitle = ref(materials[0]?.title ?? '')
```

Створіть `filteredMaterials` через `computed`. Матеріал потрапляє до результату, якщо його назва містить пошуковий рядок без урахування регістру й тема дорівнює вибраній. Для перевірки використовуйте значення з `activeFilters`; тема `all` пропускає будь-яку тему.

Створіть друге обчислюване значення `totalMinutes`, яке підсумовує поле `minutes` у `filteredMaterials`. Підсумок у початковому стані має виглядати так:

```
Знайдено: 6 · Загальна тривалість: 180 хв
```

Якщо список порожній, замість контейнера карток покажіть повідомлення `За заданими умовами матеріалів немає.` через `v-if` і `v-else`. Не копіюйте відфільтрований масив або підсумок до окремих `ref` і не змінюйте стан усередині getter-функцій `computed`.

Виведіть картки через `v-for`, передайте кожне поле об'єкта пропсом, а як `key` використайте `material.id`. Подію `select` обробіть у каталозі: її payload має стати новим значенням `selectedTitle`. Рядок про вибраний матеріал має показувати поточне значення `selectedTitle`. До пропса `selected` кожної картки передайте результат порівняння її назви з `selectedTitle`.

7. Взаємодія компонентів

Підключіть форму в шаблоні `MaterialCatalog` і явно обробіть обидві події:

```
<MaterialFilterForm @apply="applyFilters" @clear="clearFilters" />
```

Отже, `MaterialFilterForm` володіє введеними, але ще не застосованими значеннями, а `MaterialCatalog` — застосованими фільтрами, результатами й вибором картки. Компонент форми лише повідомляє про намір користувача та не змінює стан каталогу напряму.

В `App.vue` підключіть `MaterialCatalog`. Інтерактивний стан каталогу має лишатися всередині `MaterialCatalog`, а `App` відповідає лише за композицію сторінки.

8. Перевірка результату

Послідовно перевірте спостережувані результати:

1. Після запуску вигляд сторінки відповідає статичній заготовці, видно 6 карток і сумарну тривалість 180 хв.
2. Вибір теми `vue` не змінює список до надсилання форми; після натискання `Застосувати` лишаються 2 картки із сумарною тривалістю 65 хв.
3. Пошук `стан`, надісланий клавішею Enter, лишає матеріал «Реактивний стан» тривалістю 30 хв і не перезавантажує сторінку.
4. Пошук неіснуючого слова після надсилання форми показує повідомлення про порожній результат.
5. Кнопка картки оновлює напис про вибраний матеріал, а вибрана картка отримує акцентну рамку та напис `Обрано`.
6. Кнопка `Очистити` повертає у вихідний стан поля форми, 6 карток і 180 хв.
7. Клавіша Escape у будь-якому полі форми дає такий самий результат без помилок у консолі.

Зупиніть dev-сервер клавішами `Ctrl+C`, виконайте виробничу збірку й переконайтеся, що вона завершується без помилок:

```
npm run build
```

Зробіть знімок початкового стану та знімок після застосування фільтра або з порожнім результатом. На кожному знімку мають бути видимі адресний рядок, дані студента та відомості про середовище.

9. Публікація

Закомітьте роботу й надішліть її до репозиторію:

```
git add lab2
git commit -m "Лабораторна робота 2: каталог матеріалів"
git push
```

Що здається

Посилання на репозиторій `mjsf-mag` із Vue-проєктом у каталозі `lab2` та звіт із титульним аркушем, командами й двома знімками екрана: початкового стану та відфільтрованого або порожнього результату. Звіт завантажується до СЕЗН Moodle у форматі PDF.

Вимоги до змісту та оформлення звіту

- Vue-проєкт із компонентами MaterialCatalog, MaterialFilterForm і MaterialCard та набором із шести навчальних матеріалів
- Посилання на репозиторій із роботою в каталозі lab2
- Звіт із титульним аркушем, виконаними командами, знімком початкового стану каталогу та знімком відфільтрованого або порожнього результату
- Звіт у форматі PDF, завантажений до СЕЗН Moodle

Контрольні питання

1. Який виклик визначає DOM-контейнер для кореневого компонента Vue?
2. Яку форму компонента використовують як основну в прикладах курсу?
3. Як прочитати або змінити значення ref усередині JavaScript-коду?

Література до роботи

1. Vue.js. Швидкий старт — https://ua.vuejs.org/guide/quick-start.html
2. Vue.js. Основи реактивності — https://ua.vuejs.org/guide/essentials/reactivity-fundamentals.html
3. Vue.js. Обчислювані властивості — https://ua.vuejs.org/guide/essentials/computed.html
4. Vue.js. Обробка подій — https://ua.vuejs.org/guide/essentials/event-handling.html
5. Vue.js. Прив'язки до полів форм — https://ua.vuejs.org/guide/essentials/forms.html