Сучасні каркаси web-додатків

Сучасні Frontend фреймворки

Спільні принципи, моделі узгодження стану з DOM і критерії вибору

Лекція 1 · Лісняк Андрій Олександрович

Frontend — перекладач між людиною і системою

Людина мислить цілями

Знайти товар, оформити замовлення, побачити помилку — а не викликати API чи змінити запис у базі.

Система оперує даними

Backend виконує правила, читає і зберігає дані, але самі JSON та HTTP-відповіді ще не є інтерфейсом.

Frontend виконує переклад

Перетворює дію користувача на запит, а стан системи — на зрозумілий екран, керування і зворотний зв’язок.

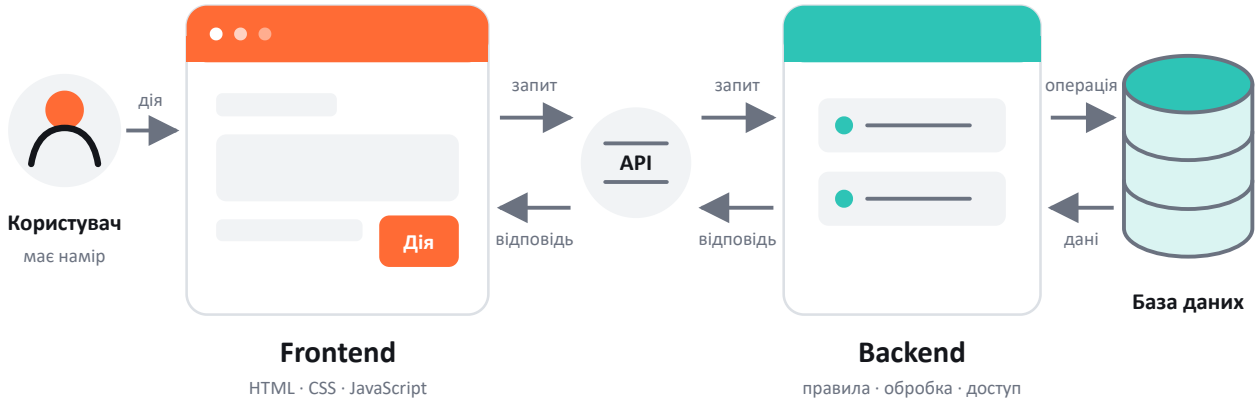
Результат — виконана задача

Користувач досягає мети, не знаючи внутрішньої будови сервера, API та сховища даних.

За матеріалами: Habr · Основи фронтенд-розробки: фреймворки, DOM-дерево, взаємодія з backend

Сучасні Frontend фреймворки

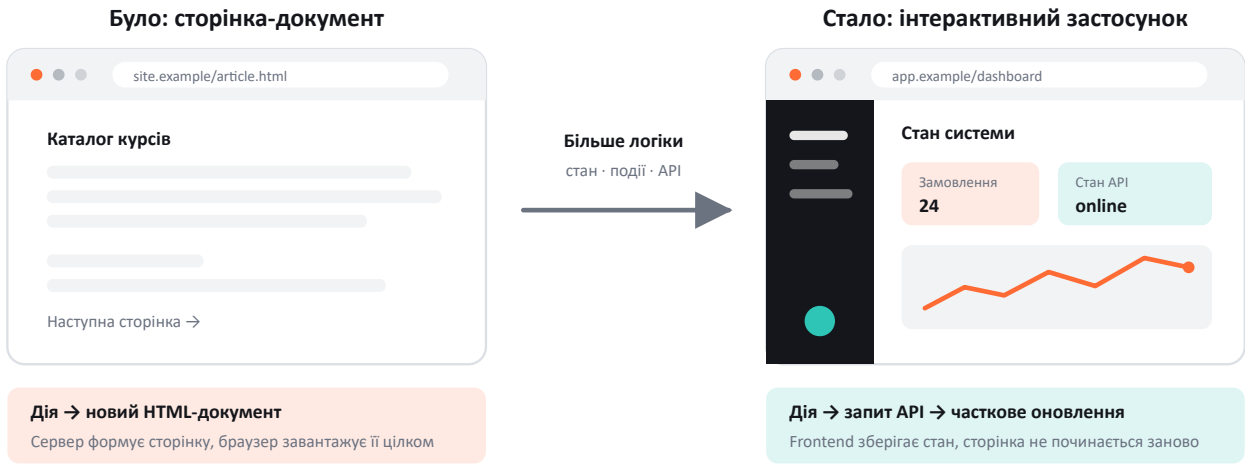
Як frontend взаємодіє з backend



Дія стає HTTP-запитом; відповідь сервера змінює стан, який frontend показує користувачеві.

Сучасні Frontend фреймворки

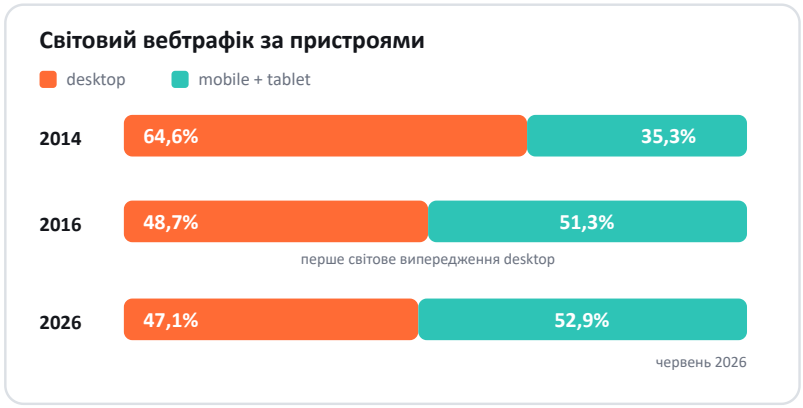
Браузер: від документа до середовища застосунку

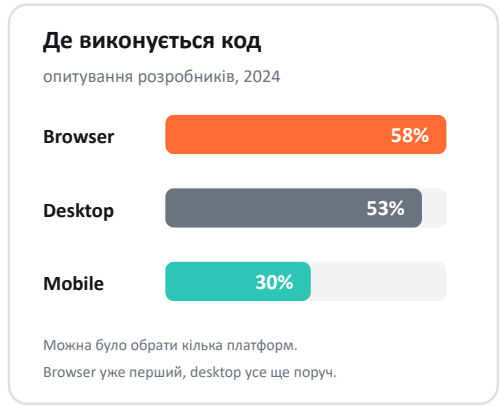


Статичні сторінки нікуди не зникли — браузер навчився ще й підтримувати довгоживучий стан та складну взаємодію.

Сучасні Frontend фреймворки

Браузер став платформою — desktop не зник



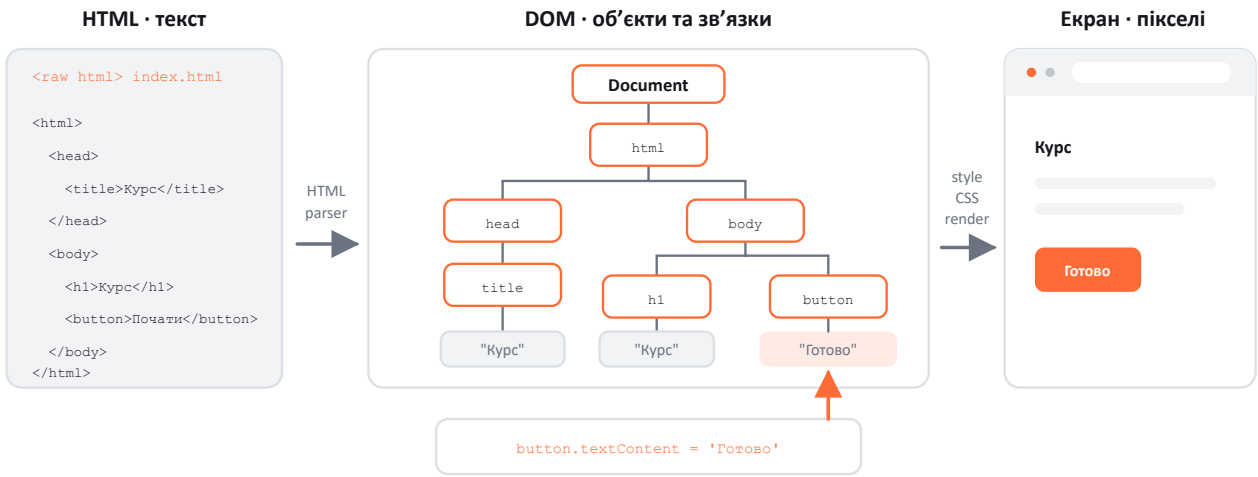


Web — не заміна одного desktop-екрана, а універсальний канал доставки інтерфейсу

Джерела: StatCounter Global Stats, 2014–2026; JetBrains Developer Ecosystem 2024, n = 23 262, множинний вибір.

Сучасні Frontend фреймворки

DOM: об’єктне дерево сторінки в пам’яті



HTML — вхідний текст, DOM — змінювана модель під час виконання, екран — результат її відтворення браузером.

Стратегії оновлення DOM



> Усі стратегії доходять до DOM API; різниця в тому, хто визначає потрібний вузол і момент оновлення.

Сучасні Frontend фреймворки

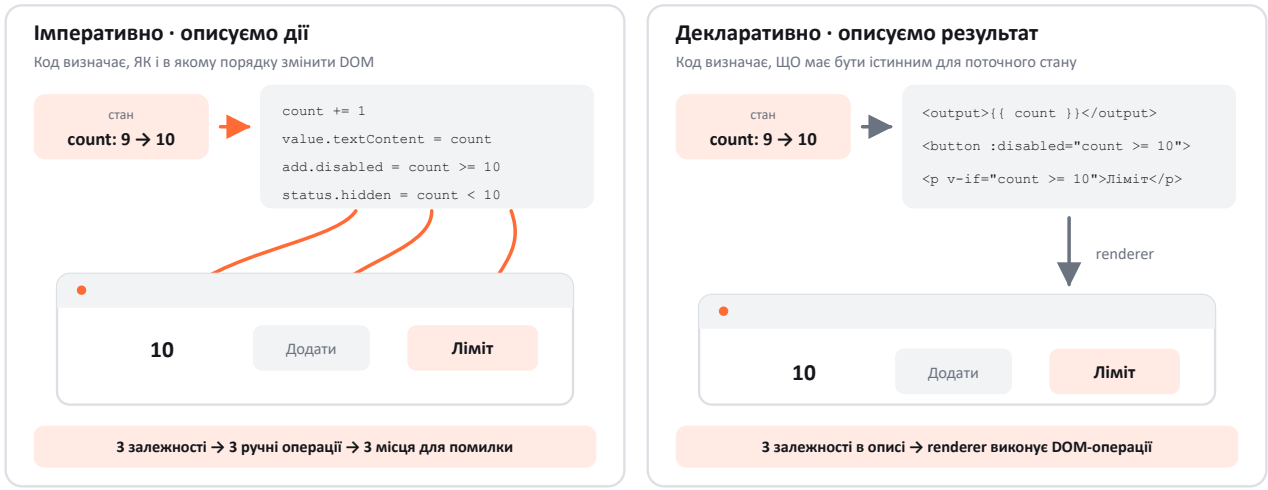
Першоджерела до стратегій оновлення DOM

Підхід	Першоджерело	Що підтверджує
Пряме керування DOM	MDN · Document Object Model ↗	Дерево вузлів-об’єктів; скрипти змінюють структуру, стиль і вміст
Virtual DOM	React · Render and Commit ↗	Render обчислює відмінності, commit застосовує мінімальні DOM-операції
Compiler-informed VDOM	Vue · Rendering Mechanism ↗	VDOM, diff, reconciliation, patch flags і підказки компілятора
Fine-grained reactivity	SolidJS · Signals ↗	Автоматичне відстеження залежностей і сповіщення лише споживачів
Точкові DOM-операції	Svelte · Virtual DOM is pure overhead ↗	Компілятор генерує оновлення для відомих динамічних місць
Incremental DOM	Google · Incremental DOM ↗	Оновлення на місці без проміжного дерева; **архівовано у 2025 році**
Change detection	Angular · Component lifecycle ↗	Перевірка шаблонів і callbacks після завершення render
Реактивні сигнали	Angular · Signals ↗	Гранулярне відстеження місць, де використано стан
Dirty checking · legacy	AngularJS 1.x · Scopes ↗	Цикл `$watch` / `$digest`; не стосується сучасного Angular

> Єдиного першоджерела для всієї класифікації немає — це узагальнення документації різних технологій.

Сучасні Frontend фреймворки

Одна зміна стану — дві моделі керування



Імперативний код задає послідовність DOM-операцій; декларативний опис задає вигляд для поточного стану.

Frontend-екосистема: 4 у фокусі + 4 поруч

Vue

Прогресивний фреймворк. Шаблони, реактивність і compiler-informed Virtual DOM.

React

UI-бібліотека. JSX, односпрямований потік даних і Virtual DOM.

Svelte

Компіляторний фреймворк. Генерує точкові DOM-операції.

Angular

Повний фреймворк. TypeScript, DI, signals і готова архітектура.

Solid

Fine-grained реактивність без Virtual DOM.

Preact

Компактна React-сумісна UI-бібліотека.

Qwik

Resumability і відкладене виконання коду.

Lit

Декларативні Web Components поверх стандартів браузера.

Джерела: vuejs.org | react.dev | svelte.dev | angular.dev | solidjs.com | preactjs.com | qwik.dev | lit.dev

Сучасні Frontend фреймворки

Порівняння 1/2 · Як працює інтерфейс

Критерій	React	Vue	Angular	Svelte	Solid
Роль	UI-бібліотека	Прогресивний framework	Повний framework	Compiler framework	Реактивна UI-бібліотека
Компоненти	JSX / TSX	HTML · SFC · JSX	HTML-шаблони	.svelte	JSX / TSX
Оновлення DOM	Render → commit	Compiler-informed VDOM	Change detection + signals	Скомпільовані операції	Fine-grained reactivity
Повторний render	Компонент + піддерево	Компонент з оптимізаціями	View checks; signals звужують роботу	Лише залежні bindings	Лише залежні обчислення
Реактивність	useState · hooks	ref · reactive	Signals · RxJS	Runes · stores	Signals · stores
Робота у браузері	VDOM + scheduler	VDOM + compiler hints	Runtime + compiler	Більше роботи під час build	Reactive runtime без VDOM
TypeScript	Дуже добра	Дуже добра	Основна мова	Дуже добра	Дуже добра
Старт навчання	Середній	Низький–середній	Високий	Низький–середній	Середній

> Джерела: Habr / RUVDS 722464 | react.dev | vuejs.org | angular.dev | svelte.dev | docs.solidjs.com

Порівняння 2/2 · Що отримує проєкт

Критерій	React	Vue	Angular	Svelte	Solid
Повнота рішення	Стек збирає команда	Core + офіційні бібліотеки	Router · HTTP · forms · DI	Core + SvelteKit	Core + SolidStart
Маршрутизація	Зовнішня / meta-framework	Офіційний Vue Router	Вбудований Router	SvelteKit	Solid Router / SolidStart
Стан	Context або бібліотеки	Pinia	Services · signals · RxJS	Runes · stores	Signals · stores
DI / контекст	Немає системної DI	provide / inject	Ієрархічна DI	Context API	Context API
SSR / SSG	Next.js та інші	Переважно Nuxt	Вбудована підтримка	SvelteKit	SolidStart
Архітектура й масштаб	Гнучко; правила команди	Помірні conventions	Строго; зручно великим командам	Мало обов'язкових conventions	Гнучко; екосистема менша
Екосистема	Найбільша	Велика	Велика корпоративна	Середня	Менша
Типові проєкти	Будь-який UI	SPA · поступова інтеграція	Великі корпоративні застосунки	Швидкі вебзастосунки	Інтерактивні UI

> Джерела: Habr / RUVDS 722464 | react.dev | vuejs.org | angular.dev | svelte.dev | docs.solidjs.com

React · свобода через композицію

Як працює

JSX декларативно описує UI. Зміна стану запускає функцію компонента; React обчислює новий результат і під час commit застосовує лише потрібні DOM-операції. Для нового production-застосунку документація радить React-фреймворк.

Сильні сторони

+ Найбільша екосистема
+ Багато готових UI-бібліотек
+ Повна програмна свобода JSX
+ Спільна модель для web і React Native
+ Багато фахівців і навчальних матеріалів

Компроміси

– Архітектуру треба визначити команді
– Routing, data fetching і state обирають окремо
– Можливі зайві повторні renders
– Hooks вимагають знання closures і dependencies
– Проєкти можуть мати дуже різну структуру

Найкраще: велика екосистема, гнучкість, web + mobile · Джерела: react.dev/learn/describing-the-ui | react.dev/learn/creating-a-react-app

Vue · поступове впровадження

Як працює

Реактивні ref і reactive живлять HTML-подібний шаблон. Компілятор створює Virtual DOM і додає patch flags, щоб runtime переважно перевіряв динамічні частини. Vue називає це Compiler-Informed Virtual DOM.

Сильні сторони

+ Зрозумілі HTML-подібні шаблони
+ Плавний старт навчання
+ Офіційні Vue Router і Pinia
+ Зручні Single-File Components
+ Можна додавати до готової HTML-сторінки
+ Options API або Composition API

Компроміси

– Два API-стилі можуть змішуватися
– Composition API має нюанси реактивності
– Екосистема менша за React
– Для full-stack SSR зазвичай обирають Nuxt

Найкраще: навчання, SPA, поступова модернізація · Джерела: vuejs.org/guide/introduction | vuejs.org/guide/extras/rendering-mechanism

Angular · готова архітектура

Як працює

Компонент поєднує TypeScript-клас, HTML-шаблон і metadata декоратора. Компоненти утворюють дерево views. Framework надає routing, HTTP, forms, signals, ієрархічну DI, CLI та SSR.

Сильні сторони

+ Узгоджена структура великого застосунку
+ TypeScript від початку
+ Повноцінна ієрархічна DI
+ Офіційні routing, forms, HTTP та SSR
+ Менше суперечок про вибір бібліотек
+ Добре для довготривалих систем

Компроміси

– Найбільше обов’язкових концепцій
– Більше шаблонного коду
– DI, RxJS і change detection ускладнюють старт
– Надмірний для малого віджета

Найкраще: великі команди й корпоративні системи · Джерела: angular.dev/overview | angular.dev/guide/components | angular.dev/guide/di

Svelte · більше роботи під час компіляції



Як працює

Декларативні HTML, CSS і JavaScript компоненти компілюються в оптимізований JavaScript. Замість повного Virtual DOM runtime отримує підготовлені оновлення. Сучасний реактивний стан описують runes — $state і $derived.

Сильні сторони

+ Компоненти близькі до HTML, CSS і JavaScript
+ Менше runtime-абстракцій
+ Зазвичай компактний клієнтський код
+ Прості transitions та animations
+ SvelteKit дає routing, SSR, SSG й endpoints

Компроміси

– Екосистема менша за React
– Власний синтаксис шаблонів і runes
– Поведінка залежить від компілятора
– Менше корпоративних інтеграцій

> Найкраще: інтерактивні сайти й малі або середні застосунки · Джерела: svelte.dev/docs/svelte/overview | svelte.dev/docs/svelte/$state

Solid · точкова реактивність



Як працює

JSX створює реальні DOM-вузли, а signals формують граф залежностей. Читання signal реєструє споживача; запис повідомляє лише залежні обчислення. Компонент переважно виконується один раз для створення view.

Сильні сторони

+ Точкові оновлення DOM
+ Низькі runtime-витрати
+ Знайомий JSX
+ Невеликий набір reactive primitives
+ Render-once mental model

Компроміси

– JSX має не React-семантику
– Менша екосистема
– Менше UI-компонентів та інтеграцій
– Менше production-досвіду на ринку

Найкраще: dashboards і UI з частими локальними оновленнями · Джерела: docs.solidjs.com/concepts/signals | github.com/solidjs/solid

Та сама задача без фреймворка

HTML і посилання на DOM · src/main.js · рядки 2–17

```
const state = { count: 0, entries: [] }


const app = document.querySelector('#app')
app.innerHTML = `
  <h1>Лічильник — без фреймворка</h1>
  <p class="count" id="count">0</p>
  <button id="add">Додати</button>
  <button id="clear">Очистити</button>
  <p id="total"></p>
  <ul id="list"></ul>`


const el = {
  count: app.querySelector('#count'),
  total: app.querySelector('#total'),
```

Ручна синхронізація · src/main.js · рядки 19–33

```
// Стан і DOM синхронізуємо вручну.
function render() {
  el.count.textContent = state.count
  el.total.textContent = `Записів: ${state.entries.length}`
  el.list.innerHTML = state.entries.length
    ? state.entries.map((e) => `<li>${e}</li>`).join('')
    : '<li class="empty">поки порожньо</li>'
}


app.querySelector('#add').addEventListener('click', () => {
  state.count += 1
  const time = new Date().toLocaleTimeString('uk-UA')
  state.entries.unshift(`Запис ${state.count} о ${time}`)
  render()
```

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

Сучасні Frontend фреймворки

Що саме тут болить

Стан у двох місцях

Об'єкт state і те, що показано в DOM, — це дві копії, які тримають синхронними руками.

Зв'язок неявний

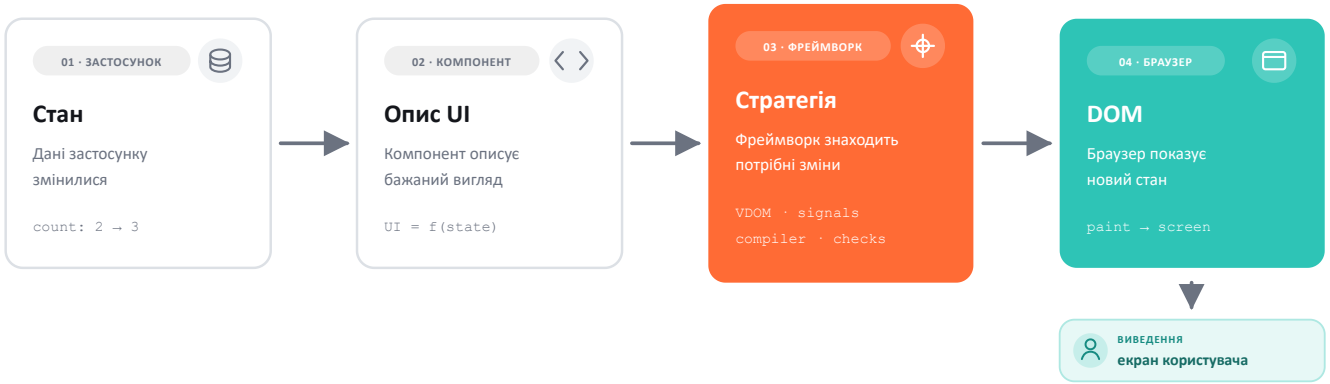
Ніщо не з'єднує дані з розміткою; з'єднання існує лише всередині render.

Зростання

Кожен новий елемент інтерфейсу — ще один рядок у render і ще одне місце, де можна забути.

> Проблема не в обсязі коду, а в тому, що правильність тримається на уважності.

Що роблять усі фреймворки



Одна мета — синхронізувати стан і DOM; фреймворки відрізняються способом знайти та застосувати зміни.

Сучасні Frontend фреймворки

Vue · reactive state → template



Реактивний стан · src/App.vue · рядки 1–17

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

const count = ref(0)
const entries = ref([])
const total = computed(() => `Записів: ${entries.value.length}`)

function add() {
  count.value += 1
  const time = new Date().toLocaleTimeString('uk-UA')
  entries.value.unshift(`Запис ${count.value} о ${time}`)
}

function clear() {
```

Декларативний шаблон · src/App.vue · рядки 20–30

```
<template>
  <h1>Лічильник — Vue</h1>
  <p class="count">{{ count }}</p>
  <button @click="add">Додати</button>
  <button @click="clear">Очистити</button>
  <p>{{ total }}</p>
  <ul>
    <li v-for="e in entries" :key="e">{{ e }}</li>
    <li v-if="!entries.length" class="empty">поки порожньо</li>
  </ul>
</template>
```

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

Сучасні Frontend фреймворки

React · state update → render → commit



State і immutable update · src/App.jsx · рядки 1–20

```jsx
import { useState } from 'react'


export default function App() {
  const [count, setCount] = useState(0)
  const [entries, setEntries] = useState([])
  const total = `Записів: ${entries.length}`


  function add() {
    const next = count + 1
    setCount(next)
    setEntries((list) => [
      `Запис ${next} о ${new Date().toLocaleTimeString('uk-UA')}`,
      ...list,
    ])
```

JSX як результат render · src/App.jsx · рядки 22–37

```jsx
  return (
    <>
      <h1>Лічильник — React</h1>
      <p className="count">{count}</p>
      <button onClick={add}>Додати</button>
      <button onClick={clear}>Очистити</button>
      <p>{total}</p>
      <ul>
        {entries.map((e) => <li key={e}>{e}</li>)}
        {entries.length === 0 && (
          <li className="empty">поки порожньо</li>
        )}
      </ul>
    </>
```

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

Сучасні Frontend фреймворки

Svelte · runes → скомпільовані bindings



Runes і звичайні присвоєння · src/App.svelte · рядки 1–16

```
<script>
  let count = $state(0)
  let entries = $state([])
  let total = $derived(`Записів: ${entries.length}`)

  function add() {
    count += 1
    const time = new Date().toLocaleTimeString('uk-UA')
    entries = [`Запис ${count} о ${time}`, ...entries]
  }

  function clear() {
    count = 0
    entries = []
```

Шаблон · src/App.svelte · рядки 18–30

```
<h1>Лічильник — Svelte</h1>
<p class="count">{count}</p>
<button onclick={add}>Додати</button>
<button onclick={clear}>Очистити</button>
<p>{total}</p>
<ul>
  {#each entries as e (e)}
    <li>{e}</li>
  {/each}
  {#if entries.length === 0}
    <li class="empty">поки порожньо</li>
  {/if}
</ul>
```

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

Angular · signals → template view



Signals і update · src/app/app.ts · рядки 21–40

```
export class App {
  protected readonly count = signal(0);
  protected readonly entries = signal<string[]>([]);
  protected readonly total = computed(
    () => `Записів: ${this.entries().length}`,
  );

  protected add(): void {
    const next = this.count() + 1;
    this.count.set(next);
    this.entries.update((list) => [
      `Запис ${next} о ${new Date().toLocaleTimeString('uk-UA')}`,
      ...list,
    ]);
```

Template control flow · src/app/app.ts · рядки 5–19

```
  template: `
    <h1>Лічильник — Angular</h1>
    <p class="count">{{ count() }}</p>
    <button (click)="add()">Додати</button>
    <button (click)="clear()">Очистити</button>
    <p>{{ total() }}</p>
    <ul>
      @for (e of entries(); track e) {
        <li>{{ e }}</li>
      }
      @if (entries().length === 0) {
        <li class="empty">поки порожньо</li>
      }
    </ul>
```

```
npm install && npm start
```

Сучасні Frontend фреймворки

Solid · signals → конкретні DOM-вузли



Signals і functional update · src/App.jsx · рядки 1–20

```
import { createMemo, createSignal, For, Show } from 'solid-js'


export default function App() {
  const [count, setCount] = createSignal(0)
  const [entries, setEntries] = createSignal([])
  const total = createMemo(() => `Записів: ${entries().length}`)


  function add() {
    const next = count() + 1
    setCount(next)
    setEntries((list) => [
      `Запис ${next} о ${new Date().toLocaleTimeString('uk-UA')}`,
      ...list,
    ])
```

JSX із реактивними читаннями · src/App.jsx · рядки 22–38

```
  return (
    <>
      <h1>Лічильник — Solid</h1>
      <p class="count">{count()}</p>
      <button onClick={add}>Додати</button>
      <button onClick={clear}>Очистити</button>
      <p>{total()}</p>
      <ul>
        <For each={entries()}>{(entry) => <li>{entry}</li>}</For>
        <Show when={entries().length === 0}>
          <li class="empty">поки порожньо</li>
        </Show>
      </ul>
    </>
```

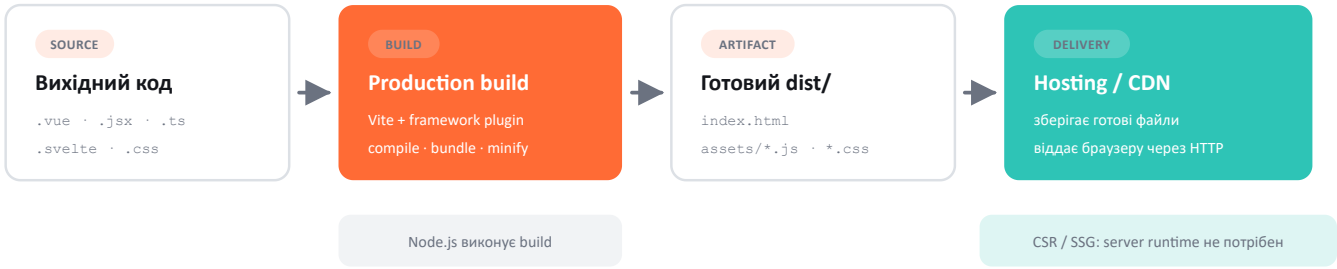
```
npm install && npm run dev
```

Ціна в готовій збірці

Підхід	JavaScript у production build (gzip)
Без фреймворка	1,61 КБ (0,87 КБ)
Solid	10,69 КБ (4,56 КБ)
Svelte	33,82 КБ (13,29 КБ)
Vue	63,48 КБ (25,31 КБ)
Angular	108,80 КБ (32,75 КБ transfer estimate)
React	193,78 КБ (61,01 КБ)

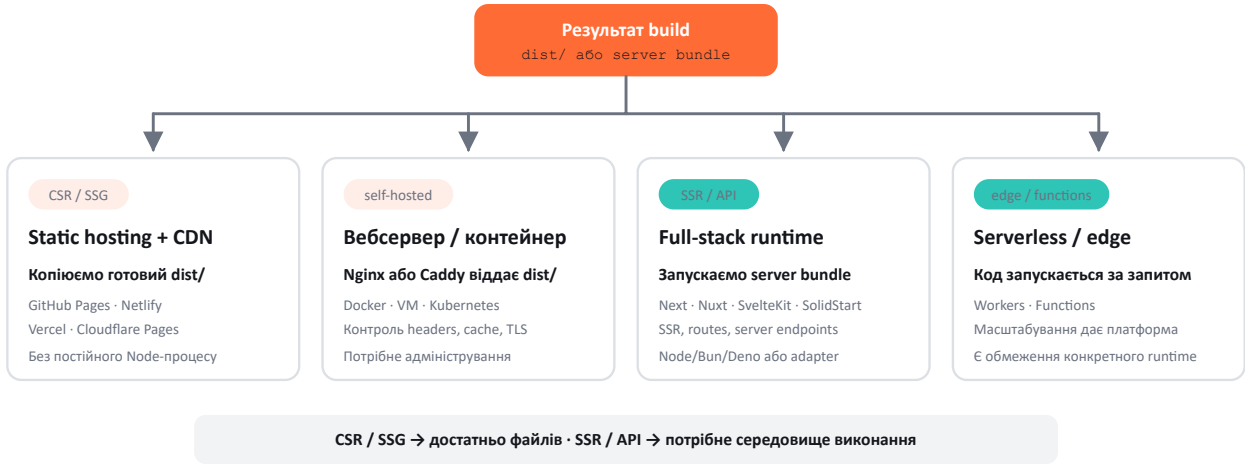
> Виміряно на шести однакових прикладах командою npm run build; кожен результат відтворюється через run.sh.

Куди дівається код перед показом



> Після збірки лишаються звичайні HTML, CSS і JavaScript. Їх треба не «запускати», а віддавати браузеру — якщо застосунок не потребує SSR.

Після build: чотири способи розгортання



Джерела: vite.dev/guide/static-deploy | nextjs.org/docs/app/guides/self-hosting | developers.cloudflare.com/pages

Дякую за увагу!

Ваші запитання