

Vue.js

Вступ



Що таке Vue

Vue (вимовляється як (англ.) /vju:/, (укр) /в'ю/) — це фреймворк, який працює працює на базі звичайного HTML, CSS та JavaScript, з можливостями декларативно програмувати користувацькі інтерфейси будь-якої складності на основі компонентів.

Vue розроблений надихаючись прикладами фреймворів React та Angular і включаючи найкращі їх особливості та підходи



Еван Ю — розробник Vue.js

<https://github.com/yyx990803>

<https://ua.vuejs.org/guide/introduction.html>

Що саме ви бачите у вказаних фрагментах?

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum">{{ num1+num2 }}</span></div>
  <div>Prod : <span id="prod">{{ num1*num2 }}</span></div>
  <button @click="onClear">Clear</button>
</div>
```

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      }
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```

Особливості Vue

- Працює на базі звичайного HTML, CSS та JavaScript (низький поріг старту)

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum"> {{ num1+num2 }} </span></div>
  <div>Prod : <span id="prod"> {{ num1*num2 }} </span></div>
  <button @click="onClear">Clear</button>
</div>
```

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      }
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```

Декларативний рендеринг: Vue розширює стандартний HTML шаблонним синтаксисом, який дозволяє нам декларативно задавати структуру HTML на основі стану описаного у JavaScript.

Особливості Vue

- Працює на базі звичайного HTML, CSS та JavaScript (низький поріг старту)
- Широко використовується компонентний підхід до розробки Single File Component (SFC)

Особливості Vue

- Працює на базі звичайного HTML, CSS та JavaScript (низький поріг старту)
- Широко використовується компонентний підхід до розробки Single File Component (SFC)

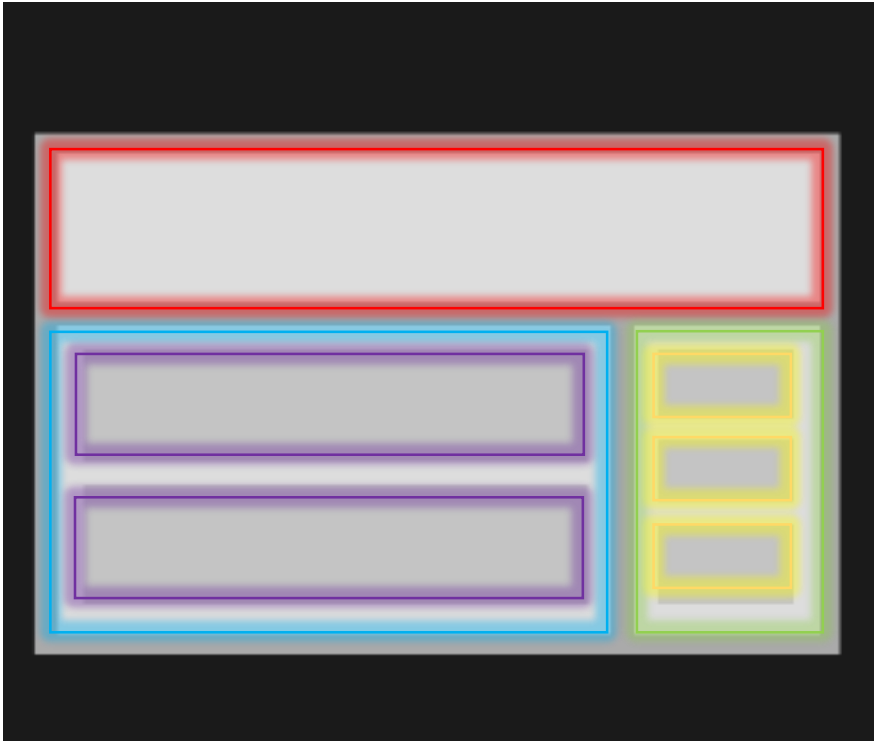


Приклад.

Потрібно розробити додаток

Особливості Vue

- Працює на базі звичайного HTML, CSS та JavaScript (низький поріг старту)
- Широко використовується компонентний підхід до розробки Single File Component (SFC)

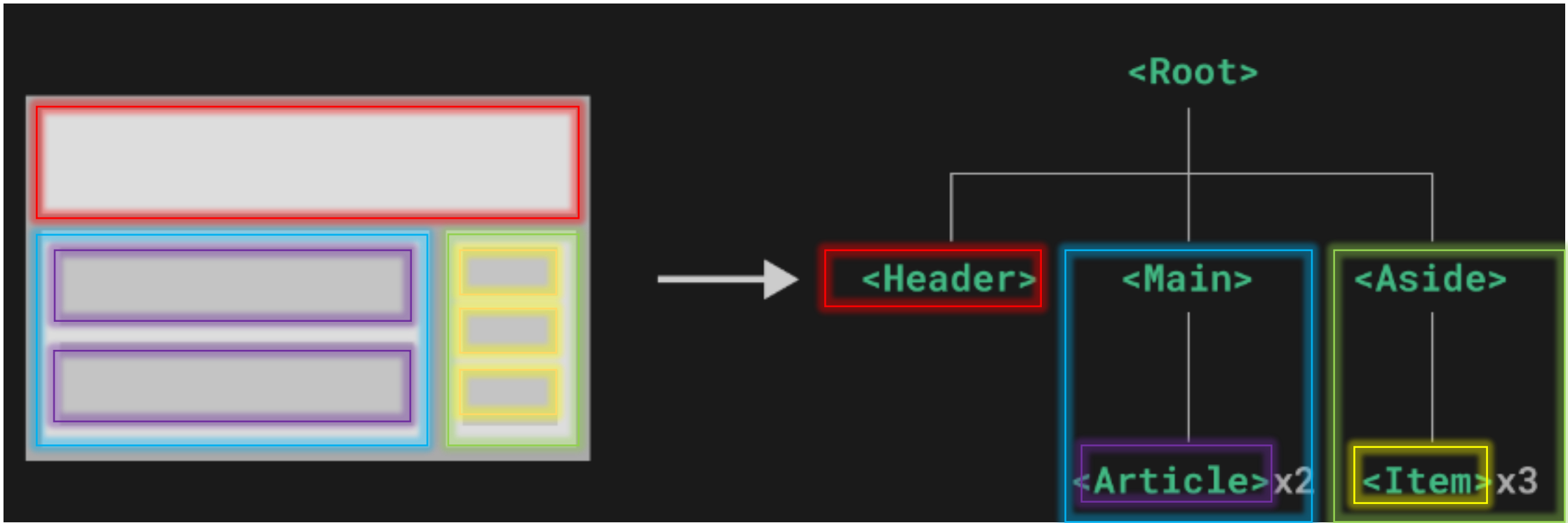


Розбиваємо на складові

Особливості Vue

- Працює на базі звичайного HTML, CSS та JavaScript (низький поріг старту)
- Широко використовується компонентний підхід до розробки Single File Component (SFC)

Описуємо окремі складові з використанням окремих компонентів



Особливості Vue

- Працює на базі звичайного HTML, CSS та JavaScript (низький поріг старту)
- Широко використовується компонентний підхід до розробки Single File Component (SFC)
- Зручний для створення одно-сторінкового додатку (Single Page Application - SPA)
- Можливість розробки додатків з рендерингом на стороні серверу (SSR)
- Генерація статичного додатку (SSG)
- Дозволяє використання у інших вже готових проєктах, розширюючи їх функціональність
- Гнучкість у використанні сторонніх рішень та компонентів
- Наявність готових інструментів та бібліотек
- Можливість розробки десктопних додатків, мобільних додатків та ін.
(<https://ua.vuejs.org/guide/extras/ways-of-using-vue.html>)
- Наявність зручної і вичерпної документації (<https://ua.vuejs.org/guide/introduction.html>)
- Велика спільнота розробників

Підходи до розробки. Стилі API

Options API

Визначаємо логіку компонента за допомогою об'єкта параметрів (опцій - options) таких як **data**, **methods**, **computed**, **mounted** та ін. (**Vue 2, Vue 3**)

Приклад застосування

```
<script>
  export default {
    // Властивості, повернуті з data(), переходять у реактивний стан
    data() {
      return {
        count: 0,
      }
    },

    // Методи — це функції, які змінюють стан і ініціюють оновлення.
    methods: {
      increment() {
        this.count++
      },
    },

    // Хуки життєвого циклу викликаються на різних етапах
    // життєвого циклу компонента.
    mounted() {
      console.log(`The initial count is ${this.count}.`)
    },
  }
}</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Composition API

Визначаємо логіку компонента за допомогою імпортованих функцій API (**тільки у Vue 3**)

Приклад застосування

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

// реактивний стан
const count = ref(0)

// функції, що змінюють стан і запускають оновлення
function increment() {
  count.value++
}

// хуки життєвого циклу
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Підходи до розробки. Стилі API

Options API

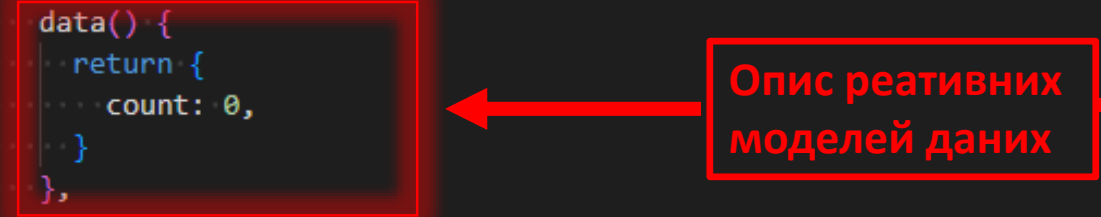
Визначаємо логіку компонента за допомогою об'єкта параметрів (опцій - options) таких як **data**, **methods**, **computed**, **mounted** та ін. (**Vue 2**, **Vue 3**)

```
<script>
export default {
  // Властивості, повернуті з data(), переходять у реактивний стан
  data() {
    return {
      count: 0,
    },
  },

  // Методи — це функції, які змінюють стан і ініціюють оновлення.
  methods: {
    increment() {
      this.count++
    },
  },

  // Хуки життєвого циклу викликаються на різних етапах
  // життєвого циклу компонента.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  },
}
</script>

<template>
<button @click="increment">Count is: {{ count }}</button>
</template>
```



Composition API

Визначаємо логіку компонента за допомогою імпортованих функцій API (**тільки у Vue 3**)

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

// реактивний стан
const count = ref(0)

// функції, що змінюють стан і запускають оновлення
function increment() {
  count.value++
}

// хуки життєвого циклу
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Підходи до розробки. Стилі API

Options API

Визначаємо логіку компонента за допомогою об'єкта параметрів (опцій - options) таких як **data**, **methods**, **computed**, **mounted** та ін. (**Vue 2**, **Vue 3**)

```
<script>
export default {
  // Властивості, повернуті з data(), переходять у реактивний стан
  data() {
    return {
      count: 0,
    },
  },

  // Методи — це функції, які змінюють стан і ініціюють оновлення.
  methods: {
    increment() {
      this.count++
    },
  },

  // Хуки життєвого циклу викликаються на різних етапах
  // життєвого циклу компонента.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  },
}
</script>

<template>
<button @click="increment">Count is: {{ count }}</button>
</template>
```

Опис методів

Composition API

Визначаємо логіку компонента за допомогою імпортованих функцій API (**тільки у Vue 3**)

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

// реактивний стан
const count = ref(0)

// функції, що змінюють стан і запускають оновлення
function increment() {
  count.value++
}

// хуки життєвого циклу
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Підходи до розробки. Стилі API

Options API

Визначаємо логіку компонента за допомогою об'єкта параметрів (опцій - options) таких як **data**, **methods**, **computed**, **mounted** та ін. (**Vue 2, Vue 3**)

```
<script>
export default {
  // Властивості, повернуті з data(), переходять у реактивний стан
  data() {
    return {
      count: 0,
    },
  },

  // Методи — це функції, які змінюють стан і ініціюють оновлення.
  methods: {
    increment() {
      this.count++
    },
  },

  // Хуки життєвого циклу викликаються на різних етапах
  // життєвого циклу компонента.
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  },
}
</script>

<template>
<button @click="increment">Count is: {{ count }}</button>
</template>
```

Хуки життєвого циклу

Composition API

Визначаємо логіку компонента за допомогою імпортованих функцій API (**тільки у Vue 3**)

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

// реактивний стан
const count = ref(0)

// функції, що змінюють стан і запускають оновлення
function increment() {
  count.value++
}

// хуки життєвого циклу
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

Для чого потрібен Vue?



Розглянемо приклад знаходження суми та добутку за введеними двома числами

First

2

Second

4

Sum : 6

Prod : 8

Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи прості засоби HTML та JS

```
<div>
  <label>
    First
    <input type="number" id="num1" value="0" onchange="changeNum1()" />
  </label>
</div>
<div>
  <label>
    Second
    <input type="number" id="num2" value="0" onchange="changeNum2()" />
  </label>
</div>
<div>Sum : <span id="sum">0</span></div>
<div>Prod : <span id="prod">0</span></div>
```

```
<script>
  let data = {
    num1: 0,
    num2: 0,
  }

  function calcResults() {
    const sum = data.num1 + data.num2
    const prod = data.num1 * data.num2
    document.getElementById('sum').innerText = sum
    document.getElementById('prod').innerText = prod
  }

  function changeNum1() {
    let num1Val = parseInt(document.getElementById('num1').value)
    data.num1 = num1Val
    calcResults()
  }

  function changeNum2() {
    let num2Val = parseInt(document.getElementById('num2').value)
    data.num2 = num2Val
    calcResults()
  }
</script>
```


Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи прості засоби HTML та JS

```
<div>
  <label>
    First
    <input type="number" id="num1" value="0" onchange="changeNum1()" />
  </label>
</div>
<div>
  <label>
    Second
    <input type="number" id="num2" value="0" onchange="changeNum2()" />
  </label>
</div>
<div>Sum : <span id="sum">0</span></div>
<div>Prod : <span id="prod">0</span></div>
```

```
<script>
  let data = {
    num1: 0,
    num2: 0,
  }
  function calcResults() {
    const sum = data.num1 + data.num2
    const prod = data.num1 * data.num2
    document.getElementById('sum').innerText = sum
    document.getElementById('prod').innerText = prod
  }
  function changeNum1() {
    let num1Val = parseInt(document.getElementById('num1').value)
    data.num1 = num1Val
    calcResults()
  }
  function changeNum2() {
    let num2Val = parseInt(document.getElementById('num2').value)
    data.num2 = num2Val
    calcResults()
  }
</script>
```

Використовуючи Proху

```
let data = {  
  num1: 0,  
  num2: 0,  
}
```

Використовуючи Proxy

Загортаємо у проху об'єкт

```
let data = {  
  num1: 0,  
  num2: 0,  
}  
  
data = new Proxy(data, {  
  get(target, prop) {  
    return target[prop]  
  },  
  set(target, prop, val) {  
    target[prop] = parseInt(val)  
    calcResults()  
  },  
})
```

Використовуючи Proxy

Загортаємо у proxy об'єкт

```
let data = {  
  num1: 0,  
  num2: 0,  
}  
  
data = new Proxy(data, {  
  get(target, prop) {  
    return target[prop]  
  },  
  set(target, prop, val) {  
    target[prop] = parseInt(val)  
    calcResults()  
  },  
})
```

```
let val = data.num1
```

Використовуючи Proxy

Загортаємо у проху об'єкт

При зчитуванні значення
викликається метод get

```
let data = {  
  num1: 0,  
  num2: 0,  
}  
  
data = new Proxy(data, {  
  get(target, prop) {  
    return target[prop]  
  },  
  set(target, prop, val) {  
    target[prop] = parseInt(val)  
    calcResults()  
  },  
})
```

let val = data . num1

Використовуючи Proxy

Загортаємо у проху об'єкт

При встановленні нового
значення викликається
метод set

```
let data = {  
  num1: 0,  
  num2: 0,  
}  
  
data = new Proxy(data, {  
  get(target, prop) {  
    return target[prop]  
  },  
  set(target, prop, val) {  
    target[prop] = parseInt(val)  
    calcResults()  
  },  
})
```

data . num1 = 7

Використовуючи Proху

First
Second
Sum : 6
Prod : 8

```
<div>
  <label>
    First
    <input
      type="number"
      id="num1"
      value="0"
      onchange="data.num1=this.value"
    />
  </label>
</div>
<div>
  <label>
    Second
    <input
      type="number"
      id="num2"
      value="0"
      onchange="data.num2=this.value"
    />
  </label>
</div>
<div>Sum : <span id="sum">0</span></div>
<div>Prod : <span id="prod">0</span></div>
```

```
<script>
  let data = {
    num1: 0,
    num2: 0,
  }

  data = new Proxy(data, {
    get(target, prop) {
      return target[prop]
    },
    set(target, prop, val) {
      target[prop] = parseInt(val)
      calcResults()
    },
  })

  function calcResults() {
    const sum = data.num1 + data.num2
    const prod = data.num1 * data.num2
    document.getElementById('sum').innerText = sum
    document.getElementById('prod').innerText = prod
  }
</script>
```

Використовуючи Proху

First

Second

Sum : 6

Prod : 8

```
<div>
  <label>
    First
    <input
      type="number"
      id="num1"
      value="0"
      onchange="data.num1=this.value"
    />
  </label>
</div>
<div>
  <label>
    Second
    <input
      type="number"
      id="num2"
      value="0"
      onchange="data.num2=this.value"
    />
  </label>
</div>
<div>Sum : <span id="sum">0</span></div>
<div>Prod : <span id="prod">0</span></div>
```

Змінюючи поле **data.num1** викликається **set**

```
<script>
  let data = {
    num1: 0,
    num2: 0,
  }

  data = new Proxy(data, {
    get(target, prop) {
      return target[prop]
    },
    set(target, prop, val) {
      target[prop] = parseInt(val)
      calcResults()
    },
  })

  function calcResults() {
    const sum = data.num1 + data.num2
    const prod = data.num1 * data.num2
    document.getElementById('sum').innerText = sum
    document.getElementById('prod').innerText = prod
  }
</script>
```


Використовуючи Proxy

First

Second

Sum : 6

Prod : 8

```
<div>
  <label>
    First
    <input
      type="number"
      id="num1"
      value="0"
      onchange="data.num1=this.value"
    />
  </label>
</div>
<div>
  <label>
    Second
    <input
      type="number"
      id="num2"
      value="0"
      onchange="data.num2=this.value"
    />
  </label>
</div>
<div>Sum : <span id="sum">0</span></div>
<div>Prod : <span id="prod">0</span></div>
```

```
<script>
  let data = {
    num1: 0,
    num2: 0,
  }

  data = new Proxy(data, {
    get(target, prop) {
      return target[prop]
    },
    set(target, prop, val) {
      target[prop] = parseInt(val)
      calcResults()
    },
  })

  function calcResults() {
    const sum = data.num1 + data.num2
    const prod = data.num1 * data.num2
    document.getElementById('sum').innerText = sum
    document.getElementById('prod').innerText = prod
  }
</script>
```

При зміні поля
data.num1
викликається
calcResult ()

First

Second

Sum : 6

Prod : 8

Використовуючи Proxy

```
<div>
  <label>
    First
    <input
      type="number"
      id="num1"
      value="0"
      onchange="data.num1=this.value"
    />
  </label>
</div>
<div>
  <label>
    Second
    <input
      type="number"
      id="num2"
      value="0"
      onchange="data.num2=this.value"
    />
  </label>
</div>
<div>Sum : <span id="sum">0</span></div>
<div>Prod : <span id="prod">0</span></div>
```

```
<script>
  let data = {
    num1: 0,
    num2: 0,
  }

  data = new Proxy(data, {
    get(target, prop) {
      return target[prop]
    },
    set(target, prop, val) {
      target[prop] = parseInt(val)
      calcResults()
    },
  })

  function calcResults() {
    const sum = data.num1 + data.num2
    const prod = data.num1 * data.num2
    document.getElementById('sum').innerText = sum
    document.getElementById('prod').innerText = prod
  }
</script>
```

Отже об'єкт data є реактивним, тобто при зміні його полів виконуємо деякі потрібні операції

При зміні поля data.num1 викликається calcResult ()

Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

Використовуючи Vue

First

Second

Sum : 6

Prod : 8

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      }
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```

*Описуємо
моделі
даних*

Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи Vue

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum">{{ num1+num2 }}</span></div>
  <div>Prod : <span id="prod">{{ num1*num2 }}</span></div>
  <button @click="onClear">Clear</button>
</div>
```

Поєднуємо моделі даних з елементами розмітки

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      },
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```

Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи Vue

Моделі даних є реактивними

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum">{{ num1+num2 }}</span></div>
  <div>Prod : <span id="prod">{{ num1*num2 }}</span></div>
  <button @click="onClear">Clear</button>
</div>
```

Поєднуємо моделі даних з елементами розмітки

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      },
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```

Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи Vue

Моделі даних є реактивними

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum">{{ num1+num2 }}</span></div>
  <div>Prod : <span id="prod">{{ num1*num2 }}</span></div>
  <button @click="onClear">Clear</button>
</div>
```

1. При зміні значення у input значення передається у об'єкт

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      }
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```

Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи Vue

Моделі даних є реактивними

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum">{{ num1+num2 }}</span></div>
  <div>Prod : <span id="prod">{{ num1*num2 }}</span></div>
  <button @click="onClear">Clear</button>
</div>
```

2. При зміні значення поля num1 його значення передається у input

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      }
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```


Розглянемо приклад знаходження суми та добутку за введеними двома числами

Для чого потрібен Vue?

First

Second

Sum : 6

Prod : 8

Використовуючи Vue

Моделі даних є реактивними

```
<div id="myApp">
  <div>
    <label>
      First
      <input type="number" id="num1" v-model="num1" />
    </label>
  </div>
  <div>
    <label>
      Second
      <input type="number" id="num2" v-model="num2" />
    </label>
  </div>
  <div>Sum : <span id="sum">{{ num1+num2 }}</span></div>
  <div>Prod : <span id="prod">{{ num1*num2 }}</span></div>
  <button @click="onClear">Clear</button>
</div>
```

3. При зміні моделей даних (num1 або num2) оновлюються інші залежні елементи розмітки

```
<script>
  const { createApp } = Vue

  const app = createApp({
    data() {
      return {
        num1: 0,
        num2: 0,
      }
    },
    methods: {
      onClear() {
        this.num1 = 0
        this.num2 = 0
      },
    },
  })

  app.mount('#myApp')
</script>
```


Загальна схема роботи звичайного динамічного сайту



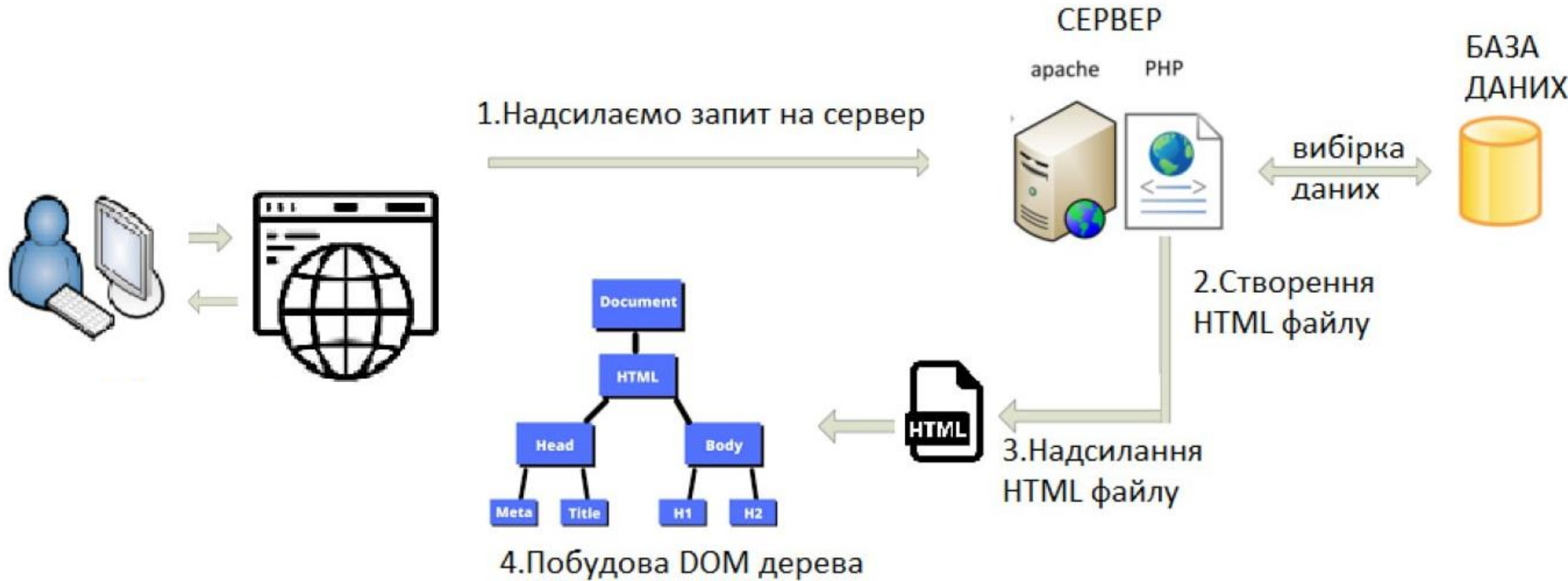
Загальна схема роботи звичайного динамічного сайту



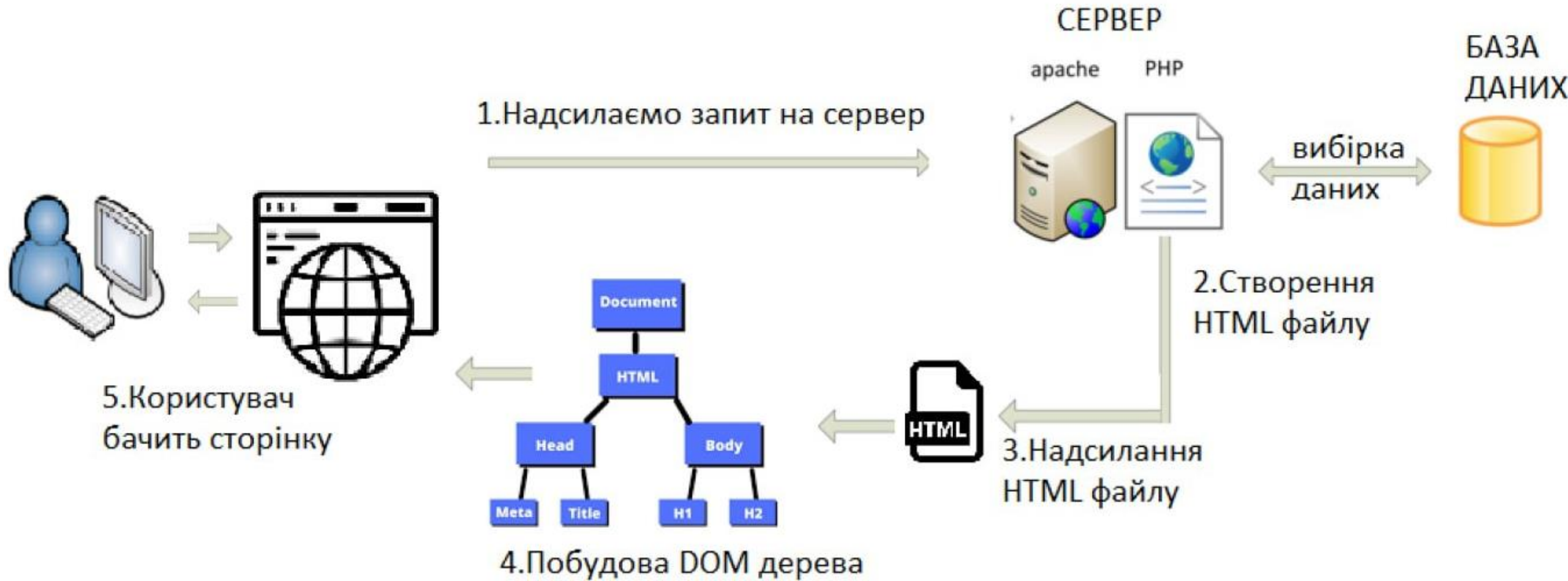
Загальна схема роботи звичайного динамічного сайту

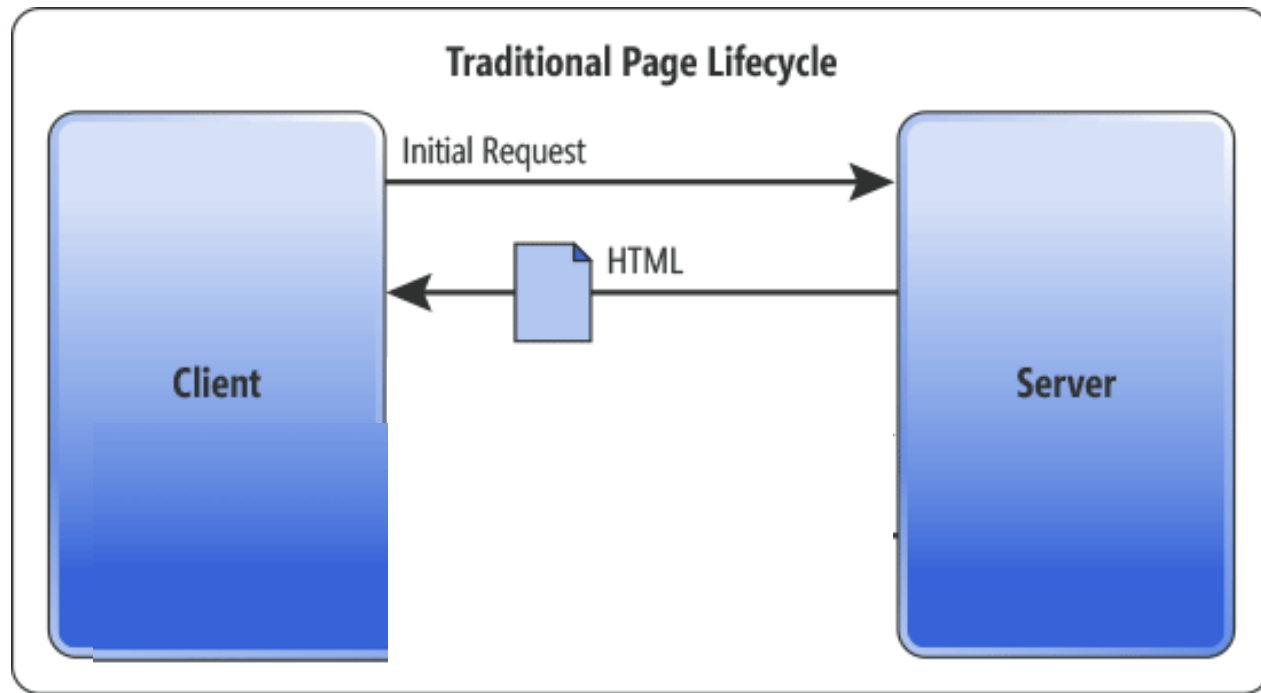


Загальна схема роботи звичайного динамічного сайту



Загальна схема роботи звичайного динамічного сайту



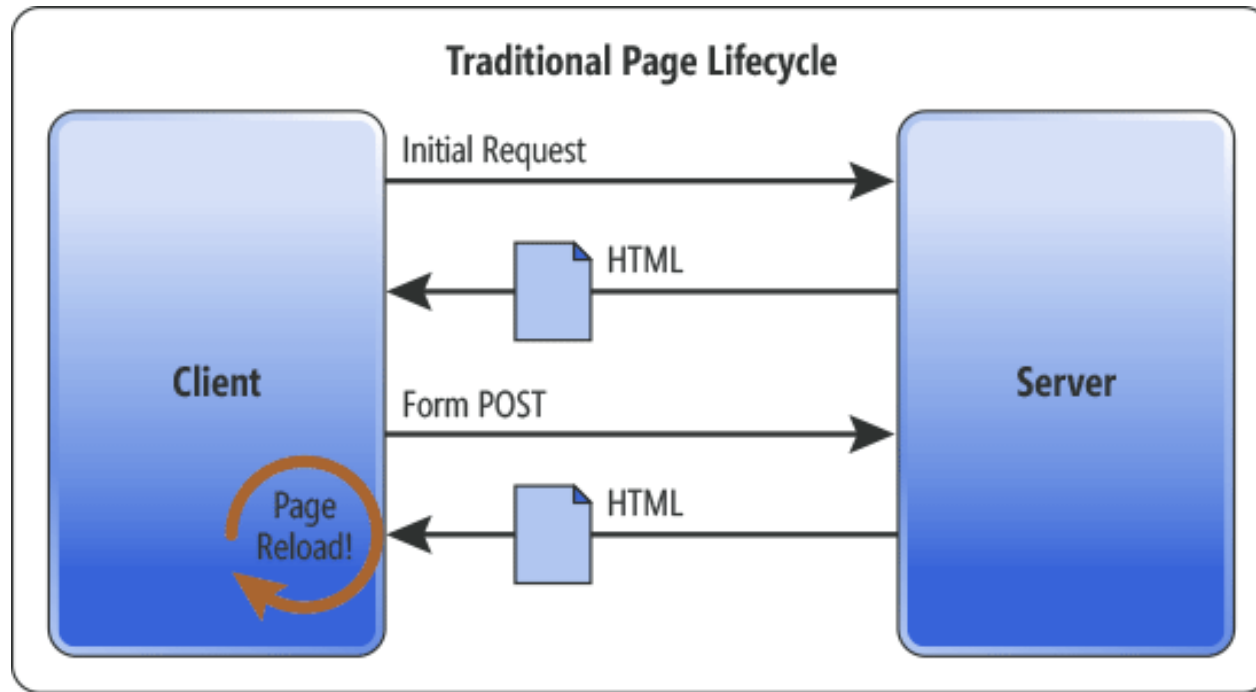


Для чого потрібен Vue?

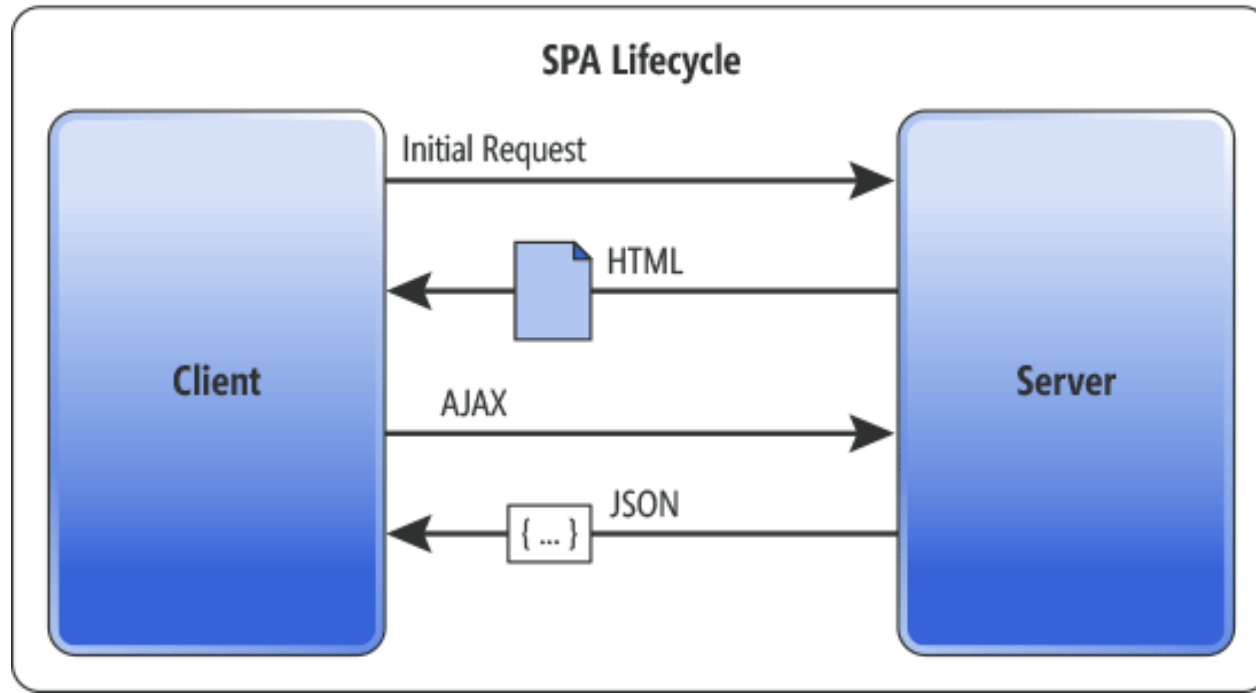
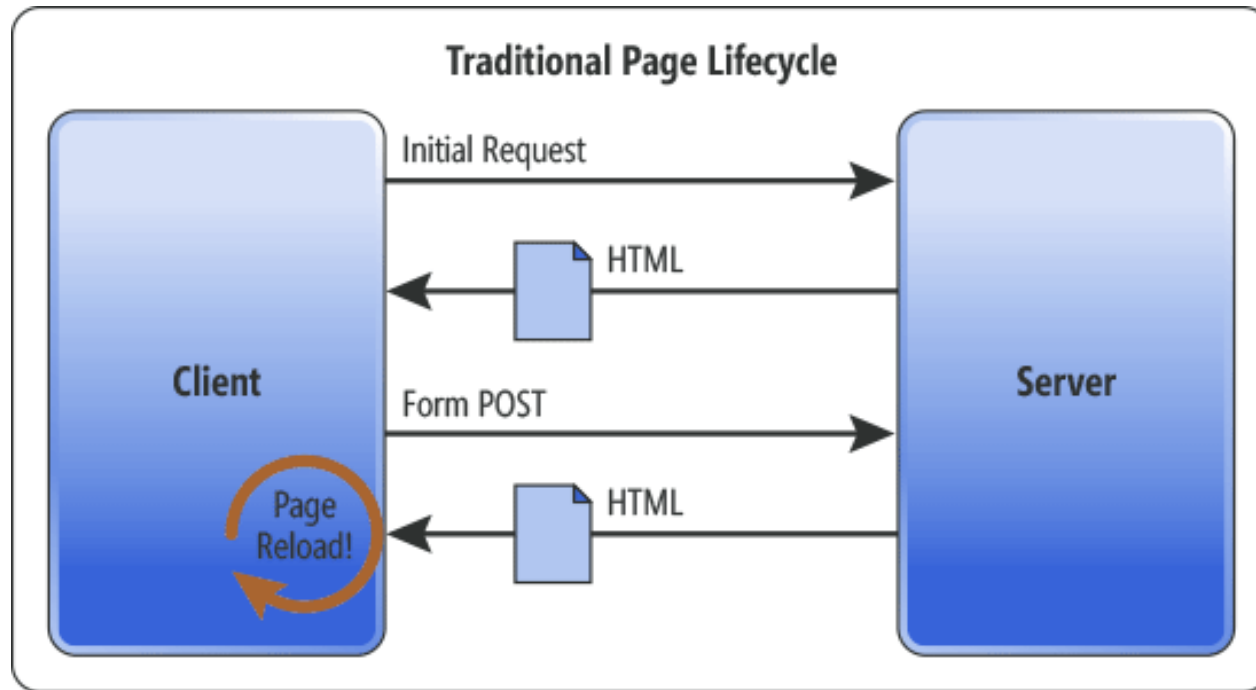
Життєвий цикл додатку

Для чого потрібен Vue?

Життєвий цикл додатку



Для чого потрібен Vue?



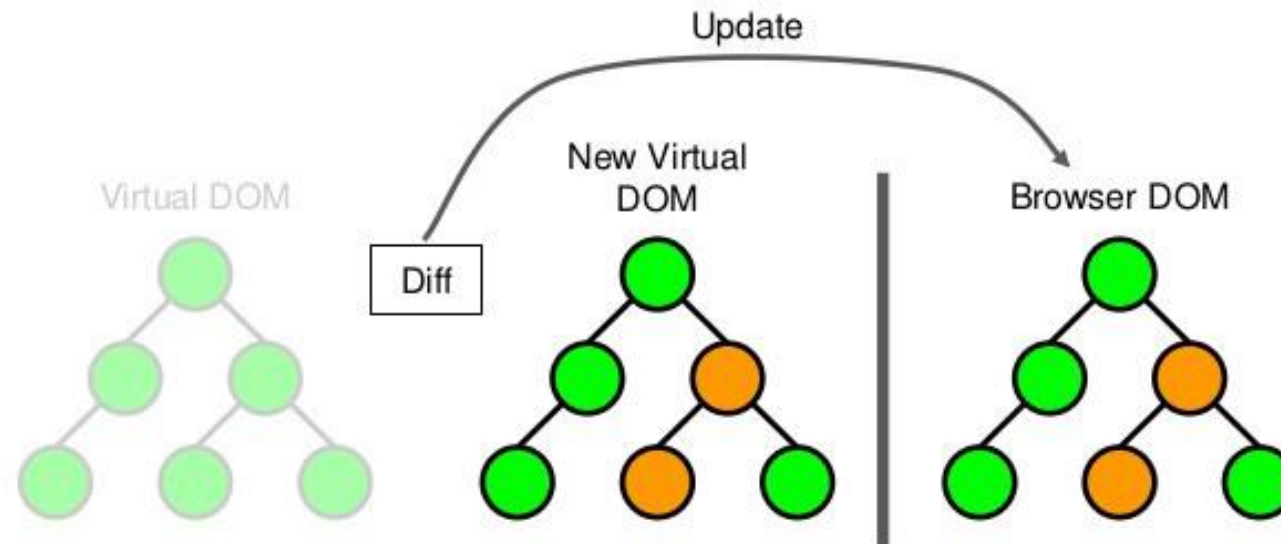
Життєвий цикл SPA додатку

Віртуальна DOM

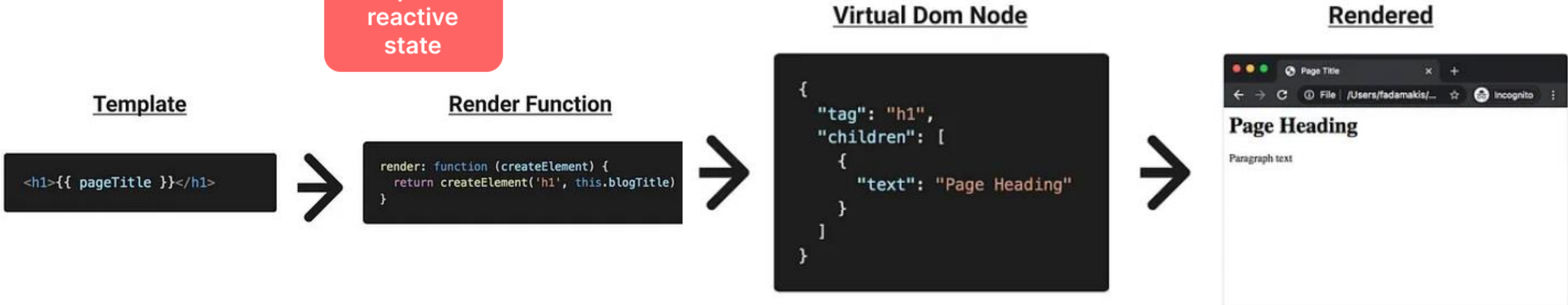
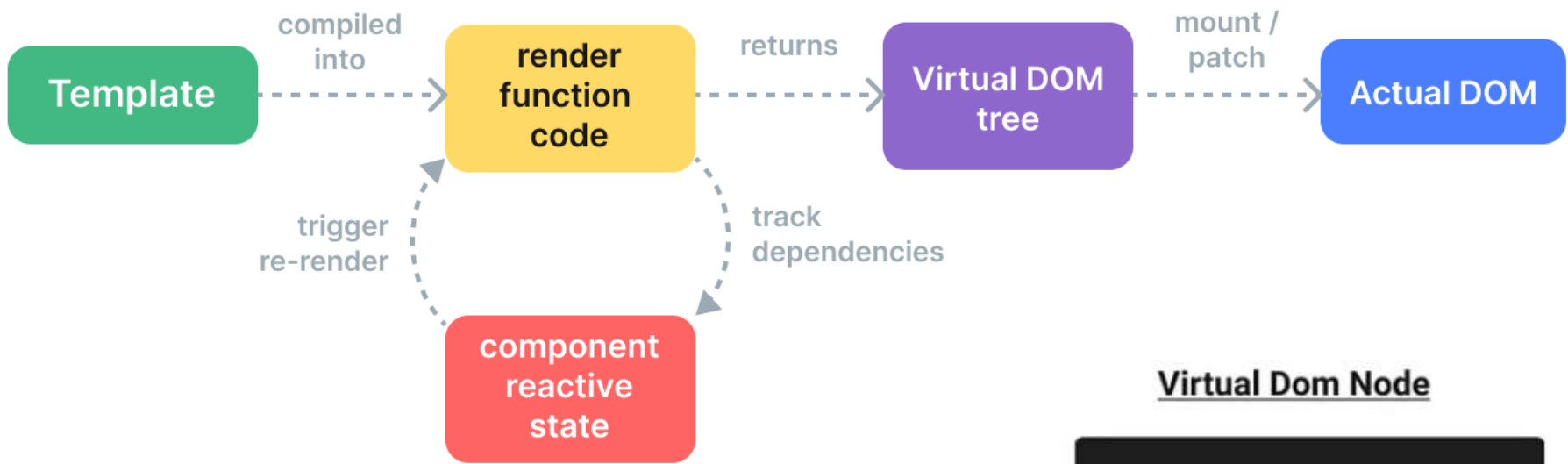
Віртуальна DOM (VDOM) — це концепція програмування, де ідеальне або "віртуальне" представлення інтерфейсу користувача зберігається в пам'яті у вигляді JavaScript об'єкта та синхронізується з "реальним" DOM.

Переваги:

- опрацювання реального дом дерева – затратна процедура, а віртуального дерева – ні, бо воно є звичайним JavaScript об'єктом
- при необхідності оновлення ми спочатку оновлюємо віртуальне дерево і порівнюючи з реальним вносимо всі необхідні зміни



Механізм рендерингу



Етапи розробки та роботи додатку на Vue

1. Розробляємо додаток на Vue
2. Збираємо додаток (з файлів Vue отримуємо JavaScript файли)
3. JavaScript файли та стартову сторінку HTML завантажуюмо на сервер
4. Користувач зайшовши на сайт завантажує стартову HTML сторінку і завантажує JavaScript файли
5. Запускається додаток і рендериться інтерфейс (далі працюють інструменти оновлення з використанням віртуального DOM дерева)