

Основи синтаксису

Розглянемо процес створення найпростішого додатку з використанням CDN посилання

1. Створюємо HTML файл, або відкриваємо вже існуючий файл, у який хочемо додати Vue елементи

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

Розглянемо процес створення найпростішого додатку з використанням CDN посилання

1. Створюємо HTML файл, або відкриваємо вже існуючий файл, у який хочемо додати Vue елементи

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

</body>
</html>
```

2. Додаємо посилання на Vue

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
</head>
<body>

</body>
</html>
```

Розглянемо процес створення найпростішого додатку з використанням CDN посилання

3. Додаємо контейнер, у якому хочемо виводити результат (наприклад, це буде div)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
</head>
<body>
  . . . . .
  <div id="app"></div>
  . . . . .
</body>
</html>
```

Розглянемо процес створення найпростішого додатку з використанням CDN посилання

4. Створюємо екземпляр додатку та монтуємо його

```
<script>
  //1. Імпортуємо createApp
  const { createApp } = Vue

  //2. Створюємо об'єкт додатку
  const app = createApp({
    . . . . .

  })

  //3. Монтуємо додаток
  app.mount('#app')
</script>
```


```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
  </head>
  <body>
    <div id="app"></div>

    <script>
      //1. Імпортуємо createApp
      const { createApp } = Vue

      //2. Створюємо об'єкт додатку
      const app = createApp({
        . . . . .

      })

      //3. Монтуємо додаток
      app.mount('#app')
    </script>
  </body>
</html>
```



Розглянемо процес створення найпростішого додатку з використанням CDN посилання

4. Додаємо моделі даних, як властивості об'єкта, що повертає функція data

```
<script>
  //1. Імпортуємо createApp
  const { createApp } = Vue

  //2. Створюємо об'єкт додатку
  const app = createApp({
    //4. Описуємо моделі даних (властивості)
    data() {
      return {
        властивість1: значення1,
        властивість2: значення2,
        . . . . .
      }
    },
  })

  //3. Монтуємо додаток
  app.mount('#app')
</script>
```

Опис моделей даних. Опція data:

- **моделі даних є реактивними** (при їх зміні автоматично оновлюються відповідні елементи розмітки та перераховуються залежні від них величини)
- **функція *data* повинна повертати об'єкт даних** (з описом реактивних моделей даних)
- **усі потрібні реактивні моделі даних повинні бути описані у *data*** (навіть якщо початкові значення null або undefined (в іншому разі вони не будуть реактивними !!!))
- **моделі даних додаються як властивості екземпляра додатку**
- **у шаблоні дані доступні лише за назвою**
- **у методах дані доступні через *this***

4. Додаємо моделі даних, як властивості об'єкта, що повертає функція data

```
<script>
  //1. Імпортуємо createApp
  const { createApp } = Vue

  //2. Створюємо об'єкт додатку
  const app = createApp({
    //4. Описуємо моделі даних (властивості)
    data() {
      return {
        message: 'Welcome!',
        price1: 100,
        price2: 150,
      }
    },
  })

  //3. Монтуємо додаток
  app.mount('#app')
</script>
```

Опис моделей даних. Опція data:

- **моделі даних є реактивними** (при їх зміні автоматично оновлюються відповідні елементи розмітки та перераховуються залежні від них величини)
- **функція *data* повинна повертати об'єкт даних** (з описом реактивних моделей даних)
- **усі потрібні реактивні моделі даних повинні бути описані у *data*** (навіть якщо початкові значення null або undefined (в іншому разі вони не будуть реактивними !!!))
- **у шаблоні дані доступні лише за назвою**
- **у методах дані доступні через *this***

Розглянемо процес створення найпростішого додатку з використанням CDN посилання

5. Використовуємо дані всередині контейнера, де змонтовано додаток

```
<div id="app">
  <h1>{{message}}</h1>
  <div>
    Product 1 price : <span>{{price1}}</span>
  </div>
  <div>
    Product 2 price : <span>{{price2}}</span>
  </div>
  <div>
    Total : <span>{{price1+price2}}</span>
  </div>
</div>
```

Інтерполяція тексту

Внутрішній контент тегів можемо задавати за схемою

<тег> **{{ дані }}** </тег>

<тег> **{{ вираз }}** </тег>

5. Використовуємо дані всередині контейнера, де змонтовано додаток

Інтерполяція тексту

Внутрішній контент можемо задавати за схемою

<тег> **{{ дані }}** </тег>

<тег> **{{ вираз }}** </тег>

У шаблоні до
даних з data
звертаємось
за назвою

```
<div id="app">
  <h1>{{message}}</h1>
  <div>
    Product 1 price : <span>{{price1}}</span>
  </div>
  <div>
    Product 2 price : <span>{{price2}}</span>
  </div>
  <div>
    Total : <span>{{price1+price2}}</span>
  </div>
</div>
```

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      message: 'Welcome!',
      price1: 100,
      price2: 150,
    }
  },
})
```

5. Використовуємо дані всередині контейнера, де змонтовано додаток

Інтерполяція тексту

Внутрішній контент можемо задавати за схемою

<тег> **{{ дані }}** </тег>

<тег> **{{ вираз }}** </тег>

```
<div id="app">
  <h1>{{message}}</h1>
  <div>
    Product 1 price : <span>{{price1}}</span>
  </div>
  <div>
    Product 2 price : <span>{{price2}}</span>
  </div>
  <div>
    Total : <span>{{price1+price2}}</span>
  </div>
</div>
```

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      message: 'Welcome!',
      price1: 100,
      price2: 150,
    }
  },
})
```

Welcome!

Product 1 price : 100
Product 2 price : 150
Total : 250

Дано ПІБ та рік народження студента. Вивести на екран

Дано два випадкові номери місяців. Вивести на екран

Розглянемо процес створення найпростішого додатку з використанням CDN посилання

6. Використовуємо дані як значення атрибутів (одностороння прив'язка: зміна даних → зміна атрибуту)

```
<div id="app">
  <h1>{{message}}</h1>
  <div>
    Product 1 price :
    <span>{{price1}}</span>
  </div>
  <div>
    Product 2 price :
    <span>{{price2}}</span>
  </div>
  <div>
    Total :
    <span>{{price1+price2}}</span>
  </div>
  
</div>
```

Значення атрибутів

Значення атрибутів задаємо за схемою:

<тег **v-bind:** **атрибут** = **"значення"**> ... </тег>

скорочена форма:

<тег **:** **атрибут** = **"значення"**> ... </тег>

Значення атрибутів

Значення атрибутів задаємо за схемою:

<тег **v-bind**: **атрибут** = "**значення**"> ... </тег>

скорочена форма:

<тег **:** **атрибут** = "**значення**"> ... </тег>

6. Використовуємо дані як значення атрибутів
(одностороння прив'язка: зміна даних → зміна атрибуту)

```
<div id="app">
  <h1>{{message}}</h1>
  <div>
    Product 1 price :
    <span>{{price1}}</span>
  </div>
  <div>
    Product 2 price :
    <span>{{price2}}</span>
  </div>
  <div>
    Total :
    <span>{{price1+price2}}</span>
  </div>
  
</div>
```

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      message: 'Welcome!',
      price1: 100,
      price2: 150,
      shopLogo: 'https://www.m-a.org.uk/resources/Shop-Logo.jpg',
      logoWidth: 100,
    },
  },
})
```

Значення атрибутів

Значення атрибутів задаємо за схемою:

<тег **v-bind**: **атрибут** = "**значення**"> ... </тег>

скорочена форма:

<тег **:** **атрибут** = "**значення**"> ... </тег>

```
<div id="app">
  <h1>{{message}}</h1>
  <div>
    Product 1 price :
    <span>{{price1}}</span>
  </div>
  <div>
    Product 2 price :
    <span>{{price2}}</span>
  </div>
  <div>
    Total :
    <span>{{price1+price2}}</span>
  </div>
  
</div>
```

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      message: 'Welcome!',
      price1: 100,
      price2: 150,
      shopLogo: 'https://www.m-a.org.uk/resources/Shop-Logo.jpg',
      logoWidth: 100,
    },
  },
})
```


Welcome!

Product 1 price : 100
Product 2 price : 150
Total : 250



7. Використовуємо дані як значення атрибутів (одностороння прив'язка: зміна даних → зміна атрибуту)

Динамічні значення назв атрибутів

```
<div id="app">
  ...
  
  <a :[attr_name]="link">News</a>
</div>
```

Динамічна назва тільки маленькими літерами !

<тег v-bind:[назва-атрибуту]= "значення"> ... </тег>

скорочена форма:

<тег :[назва-атрибуту]= "значення"> ... </тег>

7. Використовуємо дані як значення атрибутів (одностороння прив'язка: зміна даних → зміна атрибуту)

Динамічні значення назв атрибутів

Динамічні назви пишемо тільки маленькими літерами:

<тег **v-bind:** [назва-атрибуту] = "значення"> ... </тег>

скорочена форма:

<тег **:** [назва-атрибуту] = "значення"> ... </тег>

```
<div id="app">
  . . . . .
  
  <a :[attr_name]="link">News</a>
</div>
```

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      // . . . . .
      shopLogo: 'https://www.m-a.org.uk/resources/Shop-Logo.jpg',
      logoWidth: 100,
      attr_name: 'href',
      link: 'https://www.ukr.net/',
    }
  },
})
```


7. Використовуємо дані як значення атрибутів (одностороння прив'язка: зміна даних → зміна атрибуту)

Динамічні значення назв атрибутів

Динамічні назви пишемо тільки маленькими літерами:

```
<тег v-bind: [назва-атрибуту] = "значення"> ... </тег>
```

скорочена форма:

```
<тег : [назва-атрибуту] = "значення"> ... </тег>
```

```
<div id="app">
  .
  .
  .
  
  <a :[attr_name]="link">News</a>
</div>
```

```
<a href="https://www.ukr.net/">News</a>
```



```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      // . . . . .
      shopLogo: 'https://www.m-a.org.uk/resources/Shop-Logo.jpg',
      logoWidth: 100,
      attr_name: 'href',
      link: 'https://www.ukr.net/',
    },
  },
})
```

- Зміна значення в `input` → зміна значення змінної-моделі
- Зміна моделі → зміна значення в `input`

```
<div id="app">
  <label
    >Number 1
    <input type="number" v-model="num1" />
  </label>
  <label
    >Number 2
    <input type="number" v-model.number="num2" />
  </label>
  <div>Sum : <span>{{num1+num2}}</span></div>
  <div>Product : <span>{{num1*num2}}</span></div>
</div>
```

```
data() {
  return {
    num1: 2,
    num2: 5,
  }
},
```

Двостороння прив'язка даних

Значення атрибутів задаємо за схемою:

<тег **v-model** = "змінна_модель"> ... </тег>

з використанням модифікаторів (`lazy`, `number`, `trim`):

<тег **v-model** .модифікатор = "змінна_модель">
...
</тег>

- Зміна значення в input → зміна значення змінної-моделі
- Зміна моделі → зміна значення в input

```
<div id="app">
  <label>
    >Number 1
    <input type="number" v-model="num1" />
  </label>
  <label>
    >Number 2
    <input type="number" v-model.number="num2" />
  </label>
  <div>Sum : <span>{{num1+num2}}</span></div>
  <div>Product : <span>{{num1*num2}}</span></div>
</div>
```

Двостороння прив'язка даних

Значення атрибутів задаємо за схемою:

`<тег v-model = "змінна_модель"> ... </тег>`

з використанням модифікаторів (lazy, number, trim):

`<тег v-model .модифікатор = "змінна_модель"> ... </тег>`

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      num1: 2,
      num2: 5,
    },
  },
})
```

Number 1 Number 2
 Sum : 7
 Product : 10

Розробити конвертер валют. Вводиться курс долара, та кількість гривень. Вивести кількість доларів

Опис методів. Опція *methods*

```
const app = createApp({
  // . . . . .
  methods: {
    method_1(args_list) {
      . . .
    },
    method_2(args_list) {
      . . .
    },
    // . . . . .
  },
})
```

- методи описуємо у розділі *methods*
- не використовуємо стрілкових функцій !
- для звертання до даних та методів екземпляру додатку використовуємо *this*
- викликається кожного разу, якщо змінюються реактивні дані, які фігурують у функції

8. Опис методів

```
<div id="app">
  <div>
    Number 1 :
    <span>{{num1}}</span>
  </div>
  <div>
    Number 2 :
    <span>{{num2}}</span>
  </div>
  <div>
    Summ :
    <span>{{getSum()}}</span>
  </div>
</div>
```

Number 1 : 2
Number 2 : 5
Summ : 7

Опис методів. Опція *methods*

- методи описуємо у розділі *methods*
- не використовуємо стрілкових функцій !
- для звертання до даних та методів екземпляру додатку використовуємо *this*
- викликається кожного разу, якщо змінюються реактивні дані, які фігурують у функції

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      num1: 2,
      num2: 5,
    },
    methods: {
      getSum() {
        return this.num1 + this.num2
      },
    },
  },
})
```

Задано початок та кінець діапазону. Випадковим чином генерувати значення з вказаного діапазону

Обчислювальні властивості. Опція computed

Загальна форма опису

```
const app = createApp({
  computed: {
    property_1() {
      return computed_value_1
    },
    // .....

    property_2: {
      get(){
        return computed_value_2
      },
      set(newValue){
        // .....
        // опрацьовуємо нове значення newValue
        // .....
      }
    },
    // .....
  },
})
```

- обчислювальні властивості описуємо у розділі computed
- значення обчислюється на основі інших даних, можливо реактивних (з розділів data або computed)
- для звертання до даних та методів екземпляру додатку використовуємо `this`
- значення перераховується кожного разу, коли змінюються реактивні дані, на основі яких обчислюється значення цієї властивості
- значення хешуються (один раз обчислили і запам'ятали, якщо реактивні змінні, на основі яких обчислюється значення не змінилися, то наступного разу при звертанні буде повернуто збережене значення)
- можуть описуватися як функції, що обов'язково повертають значення (значення інших властивостей можна тільки зчитувати, зміна заборонена !!!)
- можуть описуватись як об'єкти з методами `get`, `set` (у методі set допускається зміна інших даних)

Обчислювальні властивості. Опція computed

Загальна форма опису

```
const app = createApp({
  computed: {
    property_1() {
      return computed_value_1
    },
    // .....

    property_2: {
      get() {
        return computed_value_2
      },
      set(newValue) {
        // .....
        // обрацьовуємо нове значення newValue
        // .....
      }
    },
    // .....
  },
})
```

Приклад

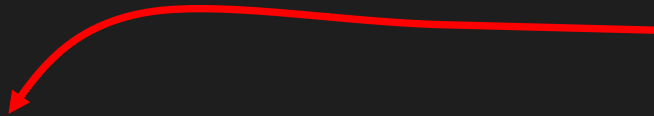
```
const app = createApp({
  data() {
    return {
      num1: 2,
      num2: 5,
    },
  },
  computed: {
    sum() {
      return this.num1 + this.num2
    },
    product() {
      return this.num1 * this.num2
    },
    // .....
  },
})
```

Обчислювальні властивості. Опція computed

Приклад використання


```
<div id="app">
  <label>
    >Number 1
    <input type="number" v-model="num1" />
  </label>
  <label>
    >Number 2
    <input type="number" v-model="num2" />
  </label>

  <div>Summ : <span>{{ sum }}</span></div>
  <div>Product : <span>{{ product }}</span></div>
</div>
```



Приклад

```
const app = createApp({
  data() {
    return {
      num1: 2,
      num2: 5,
    }
  },
  computed: {
    sum() {
      return this.num1 + this.num2
    },
    product() {
      return this.num1 * this.num2
    },
  },
})
```



Number 1 Number 2
Summ : 7
Product : 10

Обчислювальні властивості. Опція computed

- обчислювальні властивості описуємо у розділі computed
- значення обчислюється на основі інших даних, можливо реактивних (з розділів data або computed)
- для звертання до даних та методів екземпляру додатку використовуємо `this`
- значення перераховується кожного разу, коли змінюються реактивні дані, на основі яких обчислюється значення цієї властивості
- значення хешуються (один раз обчислили і запам'ятали, якщо реактивні змінні, на основі яких обчислюється значення не змінилися, то наступного разу при звертанні буде повернуто збережене значення)
- можуть описуватися як функції, що обов'язково повертають значення (значення інших властивостей можна тільки зчитувати, зміна заборонена !!!)
- можуть описуватись як об'єкти з методами `get`, `set` (у методі set допускається зміна інших даних)

```
const app = createApp({
  computed: {
    property_1() {
      return computed_value_1
    },
    // .....

    property_2: {
      get(){
        return computed_value_2
      },
      set(newValue){
        // .....
        // опрацьовуємо нове значення newValue
        // .....
      }
    },
    // .....
  },
})
```

Обчислювальні властивості

Загальна форма опису

```
const app = createApp({
  computed: {
    property_1() {
      return computed_value_1
    },
    // .....

    property_2: {
      get() {
        return computed_value_2
      },
      set(newValue) {
        // .....
        // опрацьовуємо нове значення newValue
        // .....
      }
    },
    // .....
  },
})
```

Full name

First name : Ivan

Second name : Petrov

Приклад. Вовдимо повне ім'я. Вивести окремо ім'я і прізвище

```
const app = createApp({
  data() {
    return {
      firstName: null,
      secondName: null,
    }
  },

  computed: {
    fullName: {
      get() {
        if (this.firstName && this.secondName)
          return `${this.firstName} ${this.secondName}`
        return null
      },
      set(newValue) {
        if (newValue) {
          const splitterValues = newValue.split(' ')
          if (splitterValues.length == 2) {
            this.firstName = splitterValues[0]
            this.secondName = splitterValues[1]
          }
        }
      }
    },
  },
})
```

Обчислювальні властивості.

Опція computed

Загальна форма опису

```
<div id="app">
  <label>
    >Full name
    <input type="text" v-model.lazy="fullName" />
  </label>
  <hr />
  <div>First name : <span>{{ firstName }}</span></div>
  <div>Second name : <span>{{ secondName }}</span></div>
</div>
```

Full name

First name : Ivan

Second name : Petrov

Приклад. Вовдимо повне ім'я. Вивести окремо ім'я і прізвище

```
const app = createApp({
  data() {
    return {
      firstName: null,
      secondName: null,
    }
  },

  computed: {
    fullName: {
      get() {
        if (this.firstName && this.secondName)
          return `${this.firstName} ${this.secondName}`
        return null
      },
      set(newValue) {
        if (newValue) {
          const splitterValues = newValue.split(' ')
          if (splitterValues.length == 2) {
            this.firstName = splitterValues[0]
            this.secondName = splitterValues[1]
          }
        }
      },
    },
  },
})
```

Вводиться кількість пасажирів. Вивести скільки потрібно автобусів (кожен автобус на 20 місць)

Обработка событий

Значения атрибутов задаємо за схемою:

`<тег v-on: подія = "обробник"> ... </тег>`

скорочена форма:

`<тег @подія = "обробник"> ... </тег>`

```
<div id="app">
  <div>
    Number 1 :
    <span>{{num1}}</span>
  </div>
  <div>
    Number 2 :
    <span>{{num2}}</span>
  </div>
  <button @click="getSum">Get sum</button>
  <div>
    Summ :
    <span>{{summ}}</span>
  </div>
</div>
```

Обработка событий

Значения атрибутов задаємо за схемою:

<тег v-on: подія = "обробник"> ... </тег>

скорочена форма:

<тег @подія = "обробник"> ... </тег>

```
<div id="app">
  <div>
    Number 1 :
    <span>{{num1}}</span>
  </div>
  <div>
    Number 2 :
    <span>{{num2}}</span>
  </div>
  <button @click="getSum">Get sum</button>
  <div>
    Summ :
    <span>{{summ}}</span>
  </div>
</div>
```

```
const app = createApp({
  //4. Описуємо моделі даних (властивості)
  data() {
    return {
      num1: 2,
      num2: 5,
      summ: null,
    },
  },
  methods: {
    getSum() {
      this.summ = this.num1 + this.num2
    },
  },
})
```