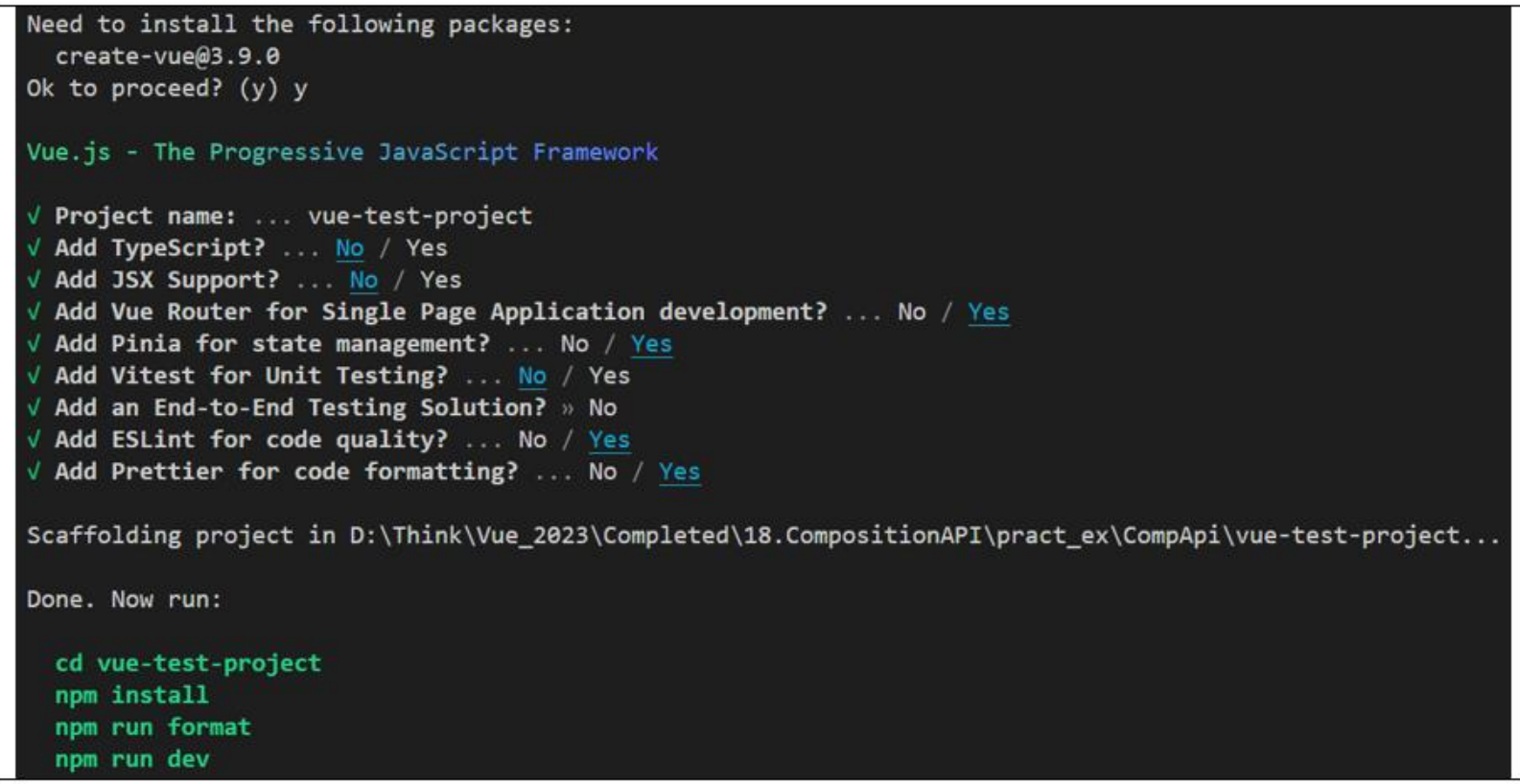
Composition API

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

https://ua.vuejs.org/api/#%D0%BA%D0%BE%D0%BC%D0%BF%D0%BE%D0%B7%D0%B8%D1%86%D1%96%D0%B9%D0%BD%D0%B8%D0%B9-api

Створення додатку

Options API	Composition API
Встановлення Vue CLI (is based on webpack) npm install -g @vue/cli # OR yarn global add @vue/cli	Встановлення (is based on Vite) **npm create vue@latest**
Створення додатку **vue create назва-додатку**	

Опис реактивних даних

Загальна форма	Приклад	Аналог у Options API

//створюється об'єкт-обгортка
// з властивістю value
import { ref } from 'vue'
 	
const посилання = ref(поч.знач.)
	import { ref } from 'vue'
	//описуємо, навіть якщо null
	const error = ref(null)
	
	//count => {value:0}
	const count = ref(0)
	
	//username => {value: 'Ivan'}
	const username = ref('Ivan')
	
	//age => {value:0}
	const age = ref(35)
	
	//book => {value:{author: 'Petro',
	// title: 'Super book' }
	// }
	let book = ref(
	  {
	  author: 'Petro',
	  title: 'Super book'
	  }
	)
		export default {
		 data() {
		  return {
		  count: 0,
		  userName:'Ivan',
		  age:35,
		  book:{
		  author: 'Petro',
		  title: 'Super book'
		  }
		  }
		 },
		}

Опис реактивних даних

Загальна форма	Приклад	Аналог у Options API

//створюється об’єкт-обгортка
// з властивістю value
import { ref } from 'vue'
 	
const посилання = ref(поч.знач.)
	import { ref } from 'vue'
	//описуємо, навіть якщо null
	const error = ref(null)
	
	//count => {value:0}
	const count = ref(0)
	
	//username => {value: ‘Ivan’}
	const username = ref(‘Ivan’)
	
	//age => {value:0}
	const age = ref(35)
	
	//book => {value:{author: 'Petro',
	// title: 'Super book' }
	// }
	let book = ref(
	  {
	  author: 'Petro',
	  title: 'Super book'
	  }
	)
		export default {
		 data() {
		  return {
		  count: 0,
		  userName:'Ivan',
		  age:35,
		  book:{
		  author: 'Petro',
		  title: 'Super book'
		  }
		  }
		 },
		}

//об’єкт перетворюється на рективний
import { reactive } from 'vue'
 	
const посилання = reactive(об’єкт
)
	import { reactive } from 'vue'
	
	let book = reactive(
	  {
	  author: 'Petro',
	  title: 'Super book'
	  }
	)	

Розробка компонентів з використанням Composition API

З використанням setup-функцій	З використанням <script setup>

<template>
. . . . . .
</template>
 	
<script>
export default {
setup() {
. . . . . .
}
}
</script>
 	
<style lang="scss" >
. . . . . .
</style>
	<template>
	. . . . . .
	</template>
	
	<script setup>
	
	. . . . . .
	
	</script>
	
	<style lang="scss" >
	. . . . . .
	</style>

Експорт реактивних даних для використання у шаблонах

З використанням setup-функцій	З використанням <script setup>	Аналог у Options API

<template>
… використання посилань …
</template>
 	
<script>
import { ref } from 'vue'
 	
export default {
// `setup` – задання стану
setup() {
. . . . опис посилань . . .
// виділення стану до шаблону
return {
… список посилань …
}
}
}
</script>
	<template>
	… використання посилань …
	</template>
	
	<script **setup**>
	import { ref } from 'vue'
	
	. . . . опис посилань . . .
	
	</script>
	
	<style lang="scss" scoped></style>
		<template>
		… використання моделей …
		</template>
		
		<script>
		export default {
		data() {
		return {
		. . . опис моделей даних . . .
		}
		},
		}
		</script>

setup-функції. Експорт реактивних даних для використання у шаблонах

Загальна форма

```
<template>
   … використання посилань …
</template>

<script>
import { ref } from 'vue'

export default {
  // `setup` – задання стану
  setup() {
    . . . . опис посилань . . .
    // виділення стану до шаблону
    return {
      … список посилань …
    }
  }
}
</script>

<style lang="scss" scoped></style>
```

Приклад

```
<template>
  <div>
    <div>Count value: {{ count }}</div>
    <div>{{ userName }}</div>
    <div>
     {{ book.title }}- {{ book.author }}
    </div>
  </div>
</template>
<script>
import { ref } from 'vue'
export default {
  // `setup` – задання стану
  setup() {
   const count = ref(0)
   let userName = ref('Ivan')
   let book = ref({
         author: 'Petro',
         title: 'Super book'
       })
    // виділення стану до шаблону
    return {
      count,
      userName,
      book
    }
  }
}
</script>
<style lang="scss" scoped></style>
```

Аналог у Options API

```
<template>
  <div>
    <div>
     Count value: {{ count + 6 }}
    </div>
    <div>{{ userName }}</div>
    <div>
     {{book.title}}-{{book.author }}
    </div>
  </div>
</template>

<script>
export default {
    data() {
        return {
            count: 0,
            userName:'Ivan',
            book:{
              author: 'Petro',
              title: 'Super book'
            }
        }
    },
}
</script>
```

<script setup>. Експорт реактивних даних для використання у шаблонах

Загальна форма

```
<template>
  … використання посилань …
</template>

<script setup>
import { ref } from 'vue'
  . . . . опис посилань . . .
</script>

<style lang="scss" scoped></style>
```

Приклад

```
<template>
  <div>
    <div>Count value: {{ count + 6
}}</div>
    <div>{{ userName }}</div>
    <div>{{ book.title }}-{{ book.author
}}</div>
  </div>
</template>

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

const count = ref(0)
let userName = ref('Ivan')
let book = ref({
      author: 'Petro',
      title: 'Super book'
    })
</script>

<style lang="scss" scoped></style>
```

Аналог у Options API

```
<template>
  <div>
    <div>
     Count value: {{ count + 6 }}
    </div>
    <div>{{ userName }}</div>
    <div>
     {{book.title}}-{{book.author }}
    </div>
  </div>
</template>

<script>
export default {
    data() {
        return {
            count: 0,
            userName:'Ivan',
            book:{
               author: 'Petro',
               title: 'Super book'
            }
        }
    },
}
</script>
```

Опис методів

З використанням setup-функцій	З використанням <script setup>	Аналог у Options API

import { ref } from 'vue'
 	
export default {
  setup() {
  //опис функції
  function назва_метода() {
  . . . . . .
  }
 	
  //виділення функції
  return {
  . . . . . .
  назва_метода
  . . . . . .
  }
  }
}
	<script ***setup***>
	  import { ref } from 'vue'
	
	  //опис функції
	  function назва_метода () {
	  . . . . . .
	  }
	</script>
		<script>
		export default {
		  methods: {
		  назва_метода () {
		  . . . . . .
		  }
		  },
		}
		</script>

Опис методів. Приклад

Загальна форма	Приклад	Аналог у Options API

<template>
 	
  Count value: {{ count }}
 	
  <button @click="*incrementCount*">
  Inc
  </button>
 	
</template>
 	
<script>
import { ref } from 'vue'
 	
export default {
  setup() {
  const count = ref(0)
 	
  function *incrementCount*() {
  count.value++
  }
 	
  // виділення стану до шаблону
  return {
  count,
  incrementCount
  }
  }
}
</script>
	<template>
	
	  Count value: {{ count }}
	
	  <button @click="*incrementCount*">
	  Inc
	  </button>
	
	</template>
	
	<script ***setup***>
	import { ref } from 'vue'
	
	const count = ref(0)
	
	function *incrementCount*() {
	  count.value++
	}
	</script>
		<template>
		
		  Count value: {{ count }}
		
		  <button @click="*incrementCount*">
		  Inc
		  </button>
		
		</template>
		
		<script>
		export default {
		  data() {
		  return {
		  count: 0,
		  }
		  },
		  methods: {
		  incrementCount() {
		  this.count++
		  },
		  },
		}
		</script>

Звертання/зміна значень у мтеодах і шаблоні

У JS методах (потрібно використовувати властивість value)	Приклад	Аналог у Options API

`import { ref } from 'vue'`
 	
`const посилання = ref( поч.знач. )`
 	
`посилання . value = нове_значення`
	`<script setup>`
	`import { ref } from 'vue'`
	`//count => { value : 0 }`
	`const count = ref(0)`
	
	`function increment() {`
	`  count.value++`
	`}`
	
	`</script>`
		`export default {`
		`  data() {`
		`  return {`
		`  count: 0,`
		`  }`
		`  },`
		
		`  methods: {`
		`  increment () {`
		`  this.count++`
		`  },`
		`  },`
		`}`
		`</script>`

У шаблоні (об’єкт розгортається, і додатково «value» вказувати не потрібно)	`<button @click="count++">`
	`  {{ count }}`
	`</button>`
		`<button @click="count++">`
		`  {{ count }}`
		`</button>`

Декілька моделей даних. Перевага CompositionAPI (посилання і функції ідуть поруч)

Загальна форма

```vue
<template>
  <div>
    <div>Count value: {{ count_1 }}</div>
    <button @click="incrementCount_1">Inc</button>

    <div>Count value: {{ count_2 }}</div>
    <button @click="incrementCount_2">Inc</button>
  </div>
</template>

<script setup>
import { ref } from 'vue'
const count_1 = ref(0)
function incrementCount_1() {
  count_1.value++
}

const count_2 = ref(0)
function incrementCount_2() {
  count_2.value++
}
</script>
```

Аналог у Options API

```vue
<template>
    <div>
        <div>Count value: {{ count_1 }}</div>
        <button @click="incrementCount_1">Inc</button>

        <div>Count value: {{ count_2 }}</div>
        <button @click="incrementCount_2">Inc</button>
    </div>
</template>

<script>
export default {
    data() {
        return {
            count_1: 0,
            count_2: 0,
        }
    },

    methods: {
        incrementCount_1() {
            this.count_1++
        },
        incrementCount_2() {
            this.count_1++
        },
    },
}
</script>
```

Обчислювальні властивості

Загальна форма	Приклад	Аналог у Options API

`import {ref, computed} from 'vue'`
 	
. . . . . .
 	
`const обч._власт = computed(`
`() => {`
. . . . . .
`return обчислене_значення`
`}`
`)`
. . . . . .
(Ctrl)
	`<template>`
	`<div>`
	`<div>Count value: {{ count }}</div>`
	`<button @click="incrementCount">`
	`Inc`
	`</button>`
	
	`<div>{{ oddOrEven }}</div>`
	`</div>`
	`</template>`
	
	`<script setup>`
	`import { ref, computed } from 'vue'`
	
	`const count = ref(0)`
	`function incrementCount() {`
	`count.value++`
	`}`
	
	`const oddOrEven = computed(() => {`
	`return count.value % 2 == 0 ?`
	`'Парне' : 'Непарне'`
	`})`
	`</script>`
		`<template>`
		`<div>`
		`<div>Count value: {{ count }}</div>`
		`<button @click="incrementCount">`
		`Inc`
		`</button>`
		
		`<div>{{ oddOrEven }}</div>`
		`</div>`
		`</template>`
		
		`<script>`
		`export default {`
		`data() {`
		`return {`
		`count: 0,`
		`}`
		`},`
		`computed: {`
		`oddOrEven() {`
		`return this.count.value % 2 == 0 ?`
		`'Парне' : 'Непарне'`
		`},`
		`},`
		
		`methods: {`
		`incrementCount() {`
		`this.count++`
		`},`
		`},`
		`}`
		`</script>`

Обчислювальні властивості з можливістю запису (з get,set)

Загальна форма	Приклад	Аналог у Options API

import{ref, **computed**} from 'vue'
 	
const обч._власт = **computed**(
  get(){
  . . . . . .
  return обчислене_значення
  },
  set(newValue){
  . . . . . .
  }
)
	<script **setup**>
	import { ref, **computed** } from 'vue'
	
	const firstName = ref('John')
	const lastName = ref('Doe')
	
	const *fullName* = **computed**({
	  // getter
	  get() {
	  return firstName.value + ' ' + lastName.value
	  },
	  // setter
	  set(newValue) {
	  [firstName.value,
	  lastName.value] =
	  newValue.split(' ')
	  }
	})
	</script>
		<script>
		export default {
		  data() {
		  return {
		  firstName: 'John',
		  lastName: 'Doe',
		  }
		  },
		  computed: {
		  fullName: {
		  // getter
		  get() {
		  return this.firstName.value + ' ' + this.lastName.value
		  },
		  // setter
		  set(newValue) {
		  ;[this.firstName.value,
		  this.lastName.value] =
		  newValue.split(' ')
		  },
		  },
		  },
		
		}

Реєстрація хуків життєвого циклу

onMounted, onUpdated, onUnmounted, onBeforeMount, onBeforeUpdate, onBeforeUnmount, onErrorCaptured, …

Загальна форма	Приклад	Аналог у Options API

import { ref,
onХук_життєвого_циклу } from 'vue'
 	
onХук_життєвого_циклу (()=>{
. . . . . .
})
	<script setup>
	import { **onMounted** } from 'vue'
	
	onMounted(() => {
	console.log(`тепер компонент змонтовано.`)
	})
	</script>
		export default {
		mounted () {
		console.log(`тепер компонент змонтовано.`)
		},
		</script>

Спостерігачі

Загальна форма	Приклад

<script setup>
import { ref, watch } from 'vue'
 	
const посилання = ref(поч.знач.)
 	
watch(
посилання,
(newValue,oldValue) => {
. . . . . .
}
)
</script>
	<template>
	
	<label>
	User age
	<input v-model="userAge"
	type="number" />
	</label>
	
	 Increase status: {{
	increaseStatus }}
	
	</template>
	
	<script setup>
	import { ref, watch } from 'vue'
	
	const userAge = ref(null)
	const increaseStatus = ref(null)
	
	watch(userAge, (newVal, oldVal) => {
	increaseStatus.value = newVal > oldVal
	})
	</script>

Спостерігачі

Загальна форма

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

const посилання = ref(поч.знач.)

 watch(
   посилання,
   (newValue,oldValue) => {
     . . . . . .
   }
 )
</script>
```

Приклад

```
<template>
  <div>
    <div>
      <label>
        User age
        <input v-model="userAge"
type="number" />
      </label>
    </div>

    <div>Increase status: {{
increaseStatus }}</div>
  </div>
</template>

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

const userAge = ref(null)
const increaseStatus = ref(null)

watch(userAge, (newVal, oldVal) => {
  increaseStatus.value = newVal > oldVal
})
</script>
```

Аналог у Options API

```
<template>
    <div>
        <div>
            <label>
                User age
                <input v-model="userAge"
type="number" />
            </label>
        </div>

        <div>Increase status: {{
increaseStatus }}</div>
    </div>
</template>

<script>
export default {
    data() {
        return {
            userAge: null,
            increaseStatus: null,
        }
    },

    watch: {
        userAge(newVal, oldVal) {
            this.increaseStatus = newVal
> oldVal
        },
    },
}
</script>
```

Вихідні типи спостерігачів

Загальна форма	Приклад

Першим аргументом watch можуть бути різні типи реактивних "джерел":
• це може бути посилання,
• обчислювані посилання,
	const x = ref(0)
	const y = ref(0)
	
	// одиночне посилання
	watch(x, (newX) => {
	  console.log(`x – це ${newX}`)
	})

• реактивний об'єкт,	//реактивний об'єкт
	const obj = reactive({ count: 0 })
	watch(
	  () => obj.count, //потрібно описувати функцію!!!
	  (count) => {
	  console.log(`кількість: ${count}`)
	  }
	)

• функція отримання,	// getter
	watch(
	  () => x.value + y.value,
	  (sum) => {
	  console.log(`сума x + y: ${sum}`)
	  }
	)

• масив із кількох джерел	// масив із кількох джерел
	watch(
	  [x, () => y.value],
	  ([newX, newY]) => {
	  console.log(`x це ${newX}, y це ${newY}`)
	})

Глибинні спостерігачі

Коли ви викликаєте watch() безпосередньо для реактивного об'єкта, це неявно створить глибинний спостерігач - зворотний виклик буде запущено для всіх вкладених змін (всіх властивостей):

```
const obj = reactive({ count: 0 })

watch(obj, (newValue, oldValue) => {
  // спрацьовує при зміні вкладених властивостей
  // Примітка: тут `newValue` дорівнюватиме `oldValue`
  // тому що вони обидва вказують на один і той же об'єкт!
})

obj.count++
```

```
<script setup>
import { reactive, watch } from 'vue'

const ob = reactive({ p1: 1, p2: 2 })

watch(ob, (ob1) => {
  console.log('----ob1')
  console.log(ob1)
})

const change = () => {
  ob.p1 = 99
}
</script>

<template>
  <div>
    <button @click="change">Go</button>
    <div>{{ ob.p1 }} - {{ ob.p2 }}</div>
  </div>
</template>
```

Негайні спостерігачі

watch за промовчанням є лінивим: зворотний виклик не буде викликано, доки не зміниться спостережуване джерело. Але в деяких випадках ми можемо захотіти, щоб та сама логіка зворотного виклику запускалася невідкладно - наприклад, ми можемо захотіти отримати деякі початкові дані, а потім повторно отримати дані щоразу, коли відповідний стан змінюється.

```
watch(
  source,
  (newValue, oldValue) => {
    // виконується негайно, а потім знову, коли `source` змінюється
  },
  { immediate: true }
)
```

watchEffect()

watchEffect() дозволяє нам негайно виконати побічний ефект, автоматично відстежуючи реактивні залежності ефекту

```
watchEffect(async () => {
  const response = await fetch(
    `https://jsonplaceholder.typicode.com/todos/${todoId.value}`
  )
  data.value = await response.json()
})
```

Тут зворотний виклик запуститься негайно, немає потреби вказувати immediate: true. Під час виконання todoId.value буде автоматично відстежуватись як залежність (подібно до обчислюваних властивостей). Щоразу,
коли todoId.value змінюється, зворотній виклик буде виконуватись.
З watchEffect() нам більше не потрібно явно вказувати todoId, як вихідне значення.

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

const a = ref(0)
const b = ref(0)
watchEffect(() => {
  console.log('---- Змінено a або b ---')
  console.log(`${a.value} - ${b.value}`)
})

const change = () => {
  a.value = 11
  b.value = 22
}
</script>

<template>
  <div>
    <button @click="change">Go</button>
  </div>
</template>
```

Приклад. Змінюються окремі посилання

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

const a = ref(0)
const b = ref(0)
watchEffect(() => {
  console.log('---- Змінено a або b ---')
  console.log(`${a.value} - ${b.value}`)
})

const change = () => {
  a.value = 11
  b.value = 22
}
</script>

<template>
  <div>
    <button @click="change">Go</button>
  </div>
</template>
```

Приклад. Змінюються властивості об’єкта

```vue
<script setup>
import { reactive, watchEffect } from 'vue'

const ob = reactive({ p1: 1, p2: 2 })

watchEffect(() => {
  console.log('---- змінено властивість ob.p1')
  console.log(ob.p1)
})

const change = () => {
  ob.p1 = 99
}
</script>

<template>
  <div>
    <button @click="change">Go</button>
    <div>{{ ob.p1 }} - {{ ob.p2 }}</div>
  </div>
</template>
```

Посилання у шаблонах

Загальна форма	Приклад	Аналог у Options API

<script setup>
import { ref } from 'vue'
 	
// оголосити посилання на елемент
// ім'я має відповідати значенню референції шаблону
 	
const посилання = ref(null)
. . . .
</script>
 	
<template>
 	
<тег ref=" посилання " />
 	
</template>
	<script setup>
	import {ref, onMounted} from 'vue'
	
	// оголосити посилання на елемент
	// ім'я має відповідати значенню референції шаблону
	
	const input = ref(null)
	const input2 = ref(null)
	
	onMounted(() => {
	input.value.focus()
	
	input.value.value = 'ok'
	input2.value.value = 'no'
	})
	</script>
	
	<template>
	
	<input ref="input" />
	<input ref="input2" />
	
	</template>
		<script>
		
		export default {
		
		mounted() {
		this.**$refs**.input.focus()
		
		this.**$refs**.input.value = 'ok'
		this.**$refs**.input2.value = 'no'
		},
		}
		</script>
		<template>
		
		<input ref="input" />
		<input ref="input2" />
		
		</template>

Посилання у шаблонах, що всередині v-for

Загальна форма	Приклад	Аналог у Options API

<script setup>
import { ref } from 'vue'
 	
// оголосити посилання на елемент
 	
const посилання = ref([ ])
. . . . . .
 	
// звертання за посиланням
посилання.value[*індекс*]
 	
. . . . . .
 	
</script>
 	
<template>
 	
<тег ***v-for***=" . . . . ."
ref=" посилання "
/>
</template>
	<script setup>
	import { ref, onMounted } from
	'vue'
	
	const input = ref([])
	
	onMounted(() => {
	input.value[1].focus()
	
	input.value[0].value = 'ok'
	input.value[1].value = 'no'
	input.value[2].value = 'question'
	})
	</script>
	
	<template>
	
	<input ref="input"
	v-for="i in 3" :key="i" />
	
	</template>
		<script>
		
		export default {
		
		mounted() {
		this.$refs.input[1].focus()
		
		this.$refs.input[0].value = 'ok'
		this.$refs.input[1].value = 'no'
		this.$refs.input[2].value = 'question'
		},
		}
		</script>
		
		<template>
		
		<input v-for="i in 3" :key="i"
		ref="input" />
		
		</template>

Опис компонентів

Загальна форма **<script setup>**	Приклад. setup - function	Аналог у Options API

<script setup>
import { ref } from 'vue'
 	
//Опис вхідних даних-властивостей
defineProps(. . . . .)
 	
//опис посилань
const посилання = **ref**(поч.знач)
 	
//опис можливих подій
defineEmits(. . . . .)
 	
</script>
 	
<template>
. . . . .
</template>
 	
<style>
...стилі, що використовуються у компоненті...
</style>
	<script >
	export default {
	
	//Опис вхідних даних-властивостей
	props: . . . . .,
	
	//опис посилань
	const посилання = ref(поч.знач)
	
	// опис можливих подій
	emits: . . . . . .,
	
	setup(props) {
	. . . . . .
	}
	}
	</script>
	<template>
	. . . . .
	</template>
	
	<style>
	...стилі, що використовуються у компоненті...
	</style>
		<template>
		. . . Опис розмітки . . .
		</template>
		<script>
		export default {
		name: “назва компонента”
		components: {
		...реєстрація комопнентів...
		},
		props:{
		...опис вхідних параметрів...
		},
		data () {
		return {
		...моделі даних...
		}
		},
		computed: {
		...обчислювальні властивості...
		},
		watch: {
		...спостерігачі...
		},
		methods: {
		...методи компонента...
		}
		...хуки життєвого циклу...
		}
		</script>
		<style>
		...стилі, що використовуються у компоненті...
		</style>

Реєстрація компонентів. Глобальна реєстрація

Загальна форма	Приклад

`import { createApp } from 'vue'`
 	
`//імпорт компонента`
`import Компонент from ' шлях '`
 	
`const app = createApp({})`
 	
`app.component(`
`  // зареєстроване ім'я`
`  'ім'я_компоненту',`
`  // реалізація компоненту`
`  Компонент`
`)`
	`import { createApp } from 'vue'`
	
	`//імпорт компонентів`
	`import MyComponentA from './App.vue'`
	`import MyComponentB from './App.vue'`
	
	`const app = createApp({})`
	
	`app`
	`  .component('ComponentA', ComponentA)`
	`  .component('ComponentB', ComponentB)`

Реєстрація компонентів. Локальна реєстрація. <script **setup**>

Загальна форма	Приклад
<script setup> //імпорт компонента import Компонент from ' шлях ' </script> <template> < Компонент /> </template>	<script setup> import ComponentA from './ComponentA.vue' </script> <template> <ComponentA /> </template>

Реєстрація компонентів. Локальна реєстрація. setup

Загальна форма	Приклад

<script>
//імпорт компонента
import Компонент from ' шлях '
export default {
components: {
Компонент
},
setup() {
// ...
}
}
 	
</script >
<template>
 	
< Компонент />
 	
</template>
	<script >
	
	import ComponentA from './ComponentA.js'
	
	export default {
	components: {
	ComponentA
	},
	setup() {
	// ...
	}
	}
	</script >
	<template>
	
	< Компонент />
	
	</template>

Опис вхідних властивостей

Загальна форма

```
<script setup>
defineProps({
  // Основна перевірка типу
  // (`null` та `undefined` значення дозволять будь-який
тип)
  propA: Number,
  // Кілька можливих типів
  propB: [String, Number],
  // Обов'язковий рядок
  propC: {
    type: String,
    required: true
  },
  // Число зі значенням за промовчанням
  propD: {
    type: Number,
    default: 100
  },
  // Об'єкт зі значенням за промовчанням
  propE: {
    type: Object,
    // Значення за промовчанням для об’єктів або масивів
    // мають бути повернуті  фабричною функцією. Функція
    // отримує необроблені реквізити,
    // отримані компонентом як аргумент.
    default(rawProps) {
      return { message: 'Привіт' }
    }
  },
  // Спеціальна функція перевірки
  propF: {
    validator(value) {
      // Значення має відповідати одному з цих рядків
      return ['успіх', 'увага',
'небезпека'].includes(value)
    }
  },
  // Функція зі значенням за промовчанням
  propG: {
    type: Function,
    // На відміну від об’єкта чи масиву за промовчанням,
це не фабрика
    // функція - це функція, яка слугуватиме значенням за
промовчанням
    default() {
      return 'Функція за промовчанням'
    }
  }
})
</script>
```

Опис вхідних властивостей

Приклад **<script setup>**	Приклад. setup - function

<script setup>
const props = defineProps(['foo'])
 	
console.log(props.foo)
</script>
	<script >
	export default {
	  props: ['foo'],
	  setup(props) {
	  // setup() отримує реквізити як
	  // перший аргумент.
	  console.log(props.foo)
	  }
	}
	</script>

Події. <script setup>. Перевірка надсилання події

Загальна форма	Приклад

<script setup>
const emit = defineEmits({
  submit(payload) {
  // повертає `true` або `false` для вказівки
  // чи пройшла/не пройшла перевірка
  }
})
</script>
	<script setup>
	const emit = defineEmits({
	  // Без перевірки
	  click: null,
	
	  // Перевірка події надсилання
	  submit: ({ email, password }) => {
	  if (email && password) {
	  return true
	  } else {
	  console.warn('Недійсні данні події надсилання!')
	  return false
	  }
	  }})
	</script>

```vue
<script setup>
const emit = defineEmits(['ev1', 'ev2'])

function doEmit1() {
  emit('ev1', 11)
}
function doEmit2() {
  emit('ev2', 22)
}
</script>

<template>
  <div>
    <button @click="doEmit1">Do Emit 1</button>
    <button @click="doEmit2">Do Emit 2</button>
  </div>
</template>
```

Події. setup-функції

Загальна форма	Приклад

export default {
  emits: ['назав_події', …],
  setup(props, ctx) {
  ctx.emit(' назав_події ')
  }
}
	export default {
	  emits: ['inFocus', 'submit'],
	  setup(props, ctx) {
	  ctx.emit('submit')
	  }
	}

Композиційні функції

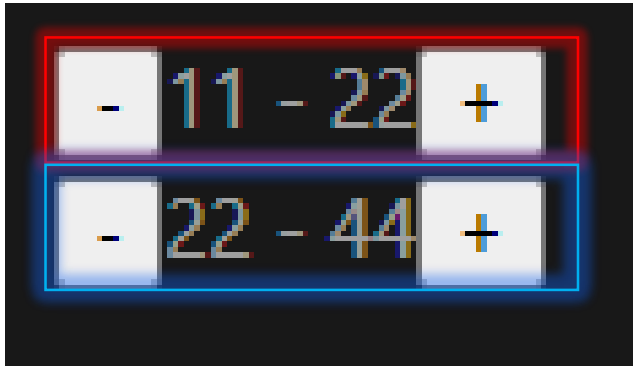
Описуємо у окремому файлі функцію, що містить посилання, функції, …
та повертаємо стан, що може бути використаний ззовні

```
// mouse.js
import { ref, onMounted, onUnmounted } from 'vue'

// за конвенцією, назви композиційних функцій починаються з "use" (англ. – використо
export function useMouse() {
  // стан, інкапсульований і керований композиційною функцією
  const x = ref(0)
  const y = ref(0)

  // композиційна функція може оновлювати свій керований стан з часом.
  function update(event) {
    x.value = event.pageX
    y.value = event.pageY
  }

  // композиційна функція також може підключитися до свого компонента-власника
  // життєвий цикл для налаштування та демонтажу побічних ефектів.
  onMounted(() => window.addEventListener('mousemove', update))
  onUnmounted(() => window.removeEventListener('mousemove', update))

  // відкрити керований стан як значення, що повертається
  return { x, y }
}
```

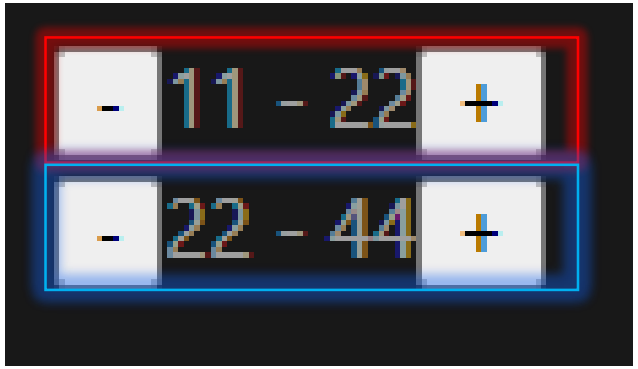
Підключаємо де потрібно

```
<script setup>
import { useMouse } from './mouse.js'

const { x, y } = useMouse()
</script>

<template>Координати миші: {{ x }}, {{ y }}</template>
```

Приклад. Потрібно створити 2 незалежних лічильники



Приклад. Потрібно створити 2 незалежних лічильники

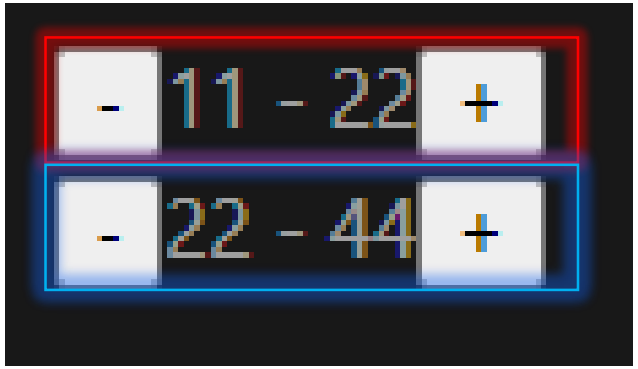



```
<template>
  <div>
    <div>
      <button @click="decrement1">-</button>
      <span>{{ counter1 }} - {{ doubleCounter1 }}</
      span>
      <button @click="increment1">+</button>
    </div>

    <div>
      <button @click="decrement2">-</button>
      <span>{{ counter2 }} - {{ doubleCounter2 }}</
      span>
      <button @click="increment2">+</button>
    </div>
  </div>
</template>
```

Приклад. Потрібно створити 2 незалежних лічильники

```vue
<script setup>
import { ref, computed } from 'vue'
//----- counter 1 -----
const counter1 = ref(11)
const doubleCounter1 = computed(() => counter1.value
* 2)
function increment1() {
  counter1.value++
}
function decrement1() {
  counter1.value--
}

//----- counter 2 -----
const counter2 = ref(22)
const doubleCounter2 = computed(() => counter2.value
* 2)
function increment2() {
  counter2.value++
}
function decrement2() {
  counter2.value--
}
</script>
```

Проблема, бо дублюється код




```vue
<template>
  <div>
    <div>
      <button @click="decrement1">-</button>
      <span>{{ counter1 }} - {{ doubleCounter1 }}</
      span>
      <button @click="increment1">+</button>
    </div>

    <div>
      <button @click="decrement2">-</button>
      <span>{{ counter2 }} - {{ doubleCounter2 }}</
      span>
      <button @click="increment2">+</button>
    </div>
  </div>
</template>
```

Приклад. Потрібно створити 2 незалежних лічильники

1. Створюємо композиційну функцію

```vue
<script setup>
import { ref, computed } from 'vue'
//---- counter 1 -----
const counter1 = ref(11)
const doubleCounter1 = computed(() => counter1.value
* 2)
function increment1() {
  counter1.value++
}
function decrement1() {
  counter1.value--
}

//---- counter 2 -----
const counter2 = ref(22)
const doubleCounter2 = computed(() => counter2.value
* 2)
function increment2() {
  counter2.value++
}
function decrement2() {
  counter2.value--
}
</script>
```

Повторюваний фрагмент виносимо у окремий файл

Приклад. Потрібно створити 2 незалежних лічильники

1. Створюємо композиційну функцію

useCounter.js views

test_1_simple > src > views > useCounter.js > useCounter > incremen

```
1   import { ref, computed } from 'vue'
2       1.----- Опис (власт., геттери, функції) ----
3   export default function useCounter(initVal = 0) {
4     const counter = ref(initVal)
5     const doubleCounter = computed(() => counter.value
      * 2)
6     function increment() {
7       counter.value++
8     }
9     function decrement() {
10      counter.value--
11    }
12
13
14
15
16
17
18  }
```

1.Повторюваний фрагмент виносимо у окремий файл з композиційною функцією

```
<script setup>
import { ref, computed } from 'vue'
//---- counter 1 -----
const counter1 = ref(11)
const doubleCounter1 = computed(() => counter1.value
* 2)
function increment1() {
  counter1.value++
}
function decrement1() {
  counter1.value--
}

//---- counter 2 -----
const counter2 = ref(22)
const doubleCounter2 = computed(() => counter2.value
* 2)
function increment2() {
  counter2.value++
}
function decrement2() {
  counter2.value--
}
</script>
```

Приклад. Потрібно створити 2 незалежних лічильники

1. Створюємо композиційну функцію

useCounter.js views

test_1_simple > src > views > useCounter.js > useCounter > incremen

```
1   import { ref, computed } from 'vue'
2       1.----- Опис (власт., геттери, функції) ----
3   export default function useCounter(initVal = 0) {
4     const counter = ref(initVal)
5     const doubleCounter = computed(() => counter.value
      * 2)
6     function increment() {
7       counter.value++
8     }
9     function decrement() {
10      counter.value--
11    }
12    return {
13      counter,
14      doubleCounter,
15      increment,
16      decrement
17    }
18  }
```

1.Повторюваний фрагмент виносимо у окремий файл з композиційною функцією

2.----- Повертаємо (власт., геттери, функції) ----

```
<script setup>
import { ref, computed } from 'vue'
//---- counter 1 -----
const counter1 = ref(11)
const doubleCounter1 = computed(() => counter1.value
* 2)
function increment1() {
  counter1.value++
}
function decrement1() {
  counter1.value--
}

//---- counter 2 -----
const counter2 = ref(22)
const doubleCounter2 = computed(() => counter2.value
* 2)
function increment2() {
  counter2.value++
}
function decrement2() {
  counter2.value--
}
</script>
```

Приклад. Потрібно створити 2 незалежних лічильники

1. Створюємо композиційну функцію

useCounter.js views

test_1_simple > src > views > useCounter.js > useCounter > incremen

```
1  import { ref, computed } from 'vue'
2
3  export default function useCounter(initVal = 0) {
4    const counter = ref(initVal)
5    const doubleCounter = computed(() => counter.value
     * 2)
6    function increment() {
7      counter.value++
8    }
9    function decrement() {
10     counter.value--
11   }
12   return {
13     counter,
14     doubleCounter,
15     increment,
16     decrement
17   }
18 }
```

1.Повторюваний фрагмент виносимо у окремий файл з композиційною функцією

```
<script setup>
import useCounter from './useCounter'
//---- counter 1 -----
const counter1 = useCounter(11)

//---- counter 2 -----
const counter2 = useCounter(22)
</script>

<template>

</template>
```

2)Використо-вуємо композиційну функцію для створення об’єктів

Приклад. Потрібно створити 2 незалежних лічильники

1. Створюємо композиційну функцію

useCounter.js views

test_1_simple > src > views > useCounter.js > useCounter > incremen

```
1   import { ref, computed } from 'vue'
2
3   export default function useCounter(initVal = 0) {
4     const counter = ref(initVal)
5     const doubleCounter = computed(() => counter.value
      * 2)
6     function increment() {
7       counter.value++
8     }
9     function decrement() {
10      counter.value--
11    }
12    return {
13      counter,
14      doubleCounter,
15      increment,
16      decrement
17    }
18  }
```

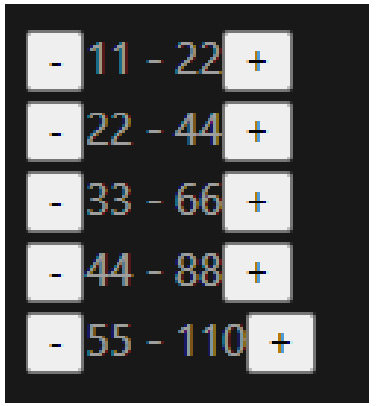
1.Повторюваний фрагмент виносимо у окремий файл з композиційною функцією

```
<script setup>
import useCounter from './useCounter'
//---- counter 1 -----
const counter1 = useCounter(11)

//---- counter 2 -----
const counter2 = useCounter(22)
</script>

<template>
  <div>
    <div>
      <button @click="counter1.decrement">-</button>
      <span>{{ counter1.counter }} - {{ counter1.
      doubleCounter }}</span>
      <button @click="counter1.increment">+</button>
    </div>

    <div>
      <button @click="counter2.decrement">-</button>
      <span>{{ counter2.counter }} - {{ counter2.
      doubleCounter }}</span>
      <button @click="counter2.increment">+</button>
    </div>
  </div>
</template>
```

2)Використо-вуємо композиційну функцію для створення об’єктів

3)Використовуємо об’єкти

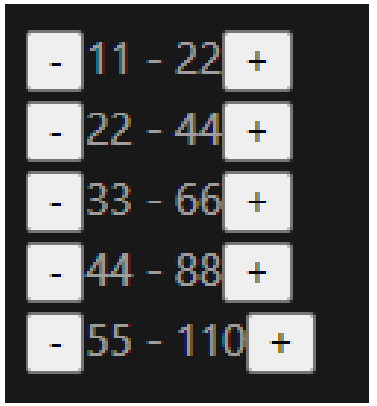
Створити список лічильників



Створити список лічильників

```vue
<script setup>
import useCounter from './useCounter'
//---- counter list -----
const initialValues = [11, 22, 33, 44, 55]
const counterList = initialValues.map((initVal) => useCounter(initVal))
</script>

<template>

</template>
```

1)Ствоюємо список об’єктів

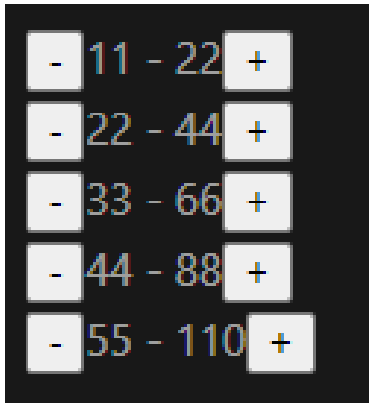


Створити список лічильників

```vue
<script setup>
import useCounter from './useCounter'
//---- counter list -----
const initialValues = [11, 22, 33, 44, 55]
const counterList = initialValues.map((initVal) => useCounter(initVal))
</script>

<template>
  <div>
    <div v-for="(counter, ind) in counterList" :key="ind">
      <button @click="counter.decrement">-</button>
      <span>{{ counter.counter }} - {{ counter.doubleCounter }}</span>
      <button @click="counter.increment">+</button>
    </div>
  </div>
</template>
```

1)Ствоюємо список об’єктів

2)Використовуємо об’єкти



Приклад. Потрібно створити 2 незалежних лічильники.

Використання деструктуризації

HomeView.vue views

test_1_simple > src > views > HomeView.vue > {} "HomeView.vue" > scr

```vue
<script setup>
import useCounter from './useCounter'
//---- counter 1 -----
const { counter: cnt1, doubleCounter: dbl1,
increment: inc1, decrement: decr1 } = useCounter(11)

//---- counter 2 -----
const { counter: cnt2, doubleCounter: dbl2,
increment: inc2, decrement: decr2 } = useCounter(22)
</script>

<template>
  <div>
    <div>
      <button @click="decr1">-</button>
      <span>{{ cnt1 }} - {{ dbl1 }}</span>
      <button @click="inc1">+</button>
    </div>
    <div>
      <button @click="decr2">-</button>
      <span>{{ cnt2 }} - {{ dbl2 }}</span>
      <button @click="inc2">+</button>
    </div>
  </div>
</template>
```

```vue
<script setup>
import useCounter from './useCounter'
//---- counter 1 -----
const counter1 = useCounter(11)

//---- counter 2 -----
const counter2 = useCounter(22)
</script>

<template>
  <div>
    <div>
      <button @click="counter1.decrement">-</button>
      <span>{{ counter1.counter }} - {{ counter1.
      doubleCounter }}</span>
      <button @click="counter1.increment">+</button>
    </div>
    <div>
      <button @click="counter2.decrement">-</button>
      <span>{{ counter2.counter }} - {{ counter2.
      doubleCounter }}</span>
      <button @click="counter2.increment">+</button>
    </div>
  </div>
</template>
```

2)Використо-вуємо композиційну функцію для створення об’єктів

3)Використовуємо об’єкти

Перевірка помилок у компонентах. onErrorCaptured

Перевірка помилок у компонентах. onErrorCaptured

Дочірній компонент, що може згенерувати помилку

TestError.vue components

mponents > TestError.vue > { } "TestError.vue" >

```vue
<template>
  <div>
    <h1>Test component</h1>
    <button @click="onClick">
      Do operation
    </button>
  </div>
</template>

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

const someObj = ref(null)

function onClick() {
  //--- призведе до помилки ---
  someObj.value.test()
}
</script>
```

Батьківський компонент, що перехоплює помилки

HomeView.vue views

test_1_simple > src > views > HomeView.vue > { } "HomeView.vue" >

```vue
<script setup>
import TestError from '@/components/TestError.vue'
import { ref, onErrorCaptured } from 'vue'



</script>

<template>
  <div>

    <test-error />
  </div>
</template>
```

Перевірка помилок у компонентах. onErrorCaptured

Дочірній компонент, що може згенерувати помилку

TestError.vue components

mponents > TestError.vue > { } "TestError.vue" >

```vue
<template>
  <div>
    <h1>Test component</h1>
    <button @click="onClick">
      Do operation
    </button>
  </div>
</template>

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

const someObj = ref(null)

function onClick() {
  //--- призведе до помилки ---
  someObj.value.test()
}
</script>
```

Батьківський компонент, що перехоплює помилки

HomeView.vue views

test_1_simple > src > views > HomeView.vue > { } "HomeView.vue" >

```vue
<script setup>
import TestError from '@/components/TestError.vue'
import { ref, onErrorCaptured } from 'vue'



</script>

<template>
  <div>

    <test-error />
  </div>
</template>
```

1. Підключаємо хук

Перевірка помилок у компонентах. onErrorCaptured

Дочірній компонент, що може згенерувати помилку

TestError.vue components

components > TestError.vue > { } "TestError.vue" >

```vue
<template>
  <div>
    <h1>Test component</h1>
    <button @click="onClick">
      Do operation
    </button>
  </div>
</template>

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

const someObj = ref(null)

function onClick() {
  //--- призведе до помилки ---
  someObj.value.test()
}
</script>
```

Батьківський компонент, що перехоплює помилки

HomeView.vue views

test_1_simple > src > views > HomeView.vue > { } "HomeView.vue" >

```vue
<script setup>
import TestError from '@/components/TestError.vue'
import { ref, onErrorCaptured } from 'vue'

const error = ref(null)

</script>

<template>
  <div>

    <test-error />
  </div>
</template>
```

1. Підключаємо хук

2. Описуємо посилання на помилку

Перевірка помилок у компонентах. onErrorCaptured

Дочірній компонент, що може згенерувати помилку

TestError.vue components

components > TestError.vue > { } "TestError.vue" >

```vue
<template>
  <div>
    <h1>Test component</h1>
    <button @click="onClick">
      Do operation
    </button>
  </div>
</template>

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

const someObj = ref(null)

function onClick() {
  //--- призведе до помилки ---
  someObj.value.test()
}
</script>
```

Батьківський компонент, що перехоплює помилки

HomeView.vue views

test_1_simple > src > views > HomeView.vue > { } "HomeView.vue" >

```vue
<script setup>
import TestError from '@/components/TestError.vue'
import { ref, onErrorCaptured } from 'vue'

const error = ref(null)

onErrorCaptured((err) => {
  error.value = err

})
</script>

<template>
  <div>

    <test-error />
  </div>
</template>
```

1. Підключаємо хук

2. Описуємо посилання на помилку

3. Перехоплюємо помилки і встановлюємо помилку

Перевірка помилок у компонентах. onErrorCaptured

Дочірній компонент, що може згенерувати помилку

TestError.vue components

mponents > TestError.vue > { } "TestError.vue" >

```vue
<template>
  <div>
    <h1>Test component</h1>
    <button @click="onClick">
      Do operation
    </button>
  </div>
</template>

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

const someObj = ref(null)

function onClick() {
  //--- призведе до помилки ---
  someObj.value.test()
}
</script>
```

Батьківський компонент, що перехоплює помилки

HomeView.vue views

test_1_simple > src > views > HomeView.vue > { } "HomeView.vue" >

```vue
<script setup>
import TestError from '@/components/TestError.vue'
import { ref, onErrorCaptured } from 'vue'

const error = ref(null)

onErrorCaptured((err) => {
  error.value = err
  return false
})
</script>

<template>
  <div>
    <test-error />
  </div>
</template>
```

1. Підключаємо хук

2. Описуємо посилання на помилку

3. Перехоплюємо помилки і встановлюємо помилку

4. Зупиняємо поширення помилки

Перевірка помилок у компонентах. onErrorCaptured

Дочірній компонент, що може згенерувати помилку

TestError.vue components

components > TestError.vue > { } "TestError.vue" >

```vue
<template>
  <div>
    <h1>Test component</h1>
    <button @click="onClick">
      Do operation
    </button>
  </div>
</template>

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

const someObj = ref(null)

function onClick() {
  //--- призведе до помилки ---
  someObj.value.test()
}
</script>
```

Батьківський компонент, що перехоплює помилки

HomeView.vue views

test_1_simple > src > views > HomeView.vue > { } "HomeView.vue" >

```vue
<script setup>
import TestError from '@/components/TestError.vue'
import { ref, onErrorCaptured } from 'vue'

const error = ref(null)

onErrorCaptured((err) => {
  error.value = err
  return false
})
</script>

<template>
  <div>
    <div v-if="error">Error</div>
    <test-error v-else />
  </div>
</template>
```

1. Підключаємо хук
2. Описуємо посилання на помилку
3. Перехоплюємо помилки і встановлюємо помилку
4. Зупиняємо поширення помилки
5. Аналізуємо помилку і відображаємо потрібні елементи чи повідомлення