

Vuex

Vuex is a **state management pattern + library** for Vue.js applications. It serves as a centralized store for all the components in an application, with rules ensuring that the state can only be mutated in a predictable fashion.

<https://vuex.vuejs.org/>

Для чого потрібе Vuex?

Розглянемо меджер задач

Додати задачу

Задача

Пріоритет

Список задач

Випити каву - 100

Випити ще одну каву - 120

Перевірити пошту - 5

Зробити зарядку - 30

Для чого потрібе Vuex?

Розглянемо меджер задач

Додати задачу

Задача
Пріоритет

— **TodoForm**

TodoManager

Список задач

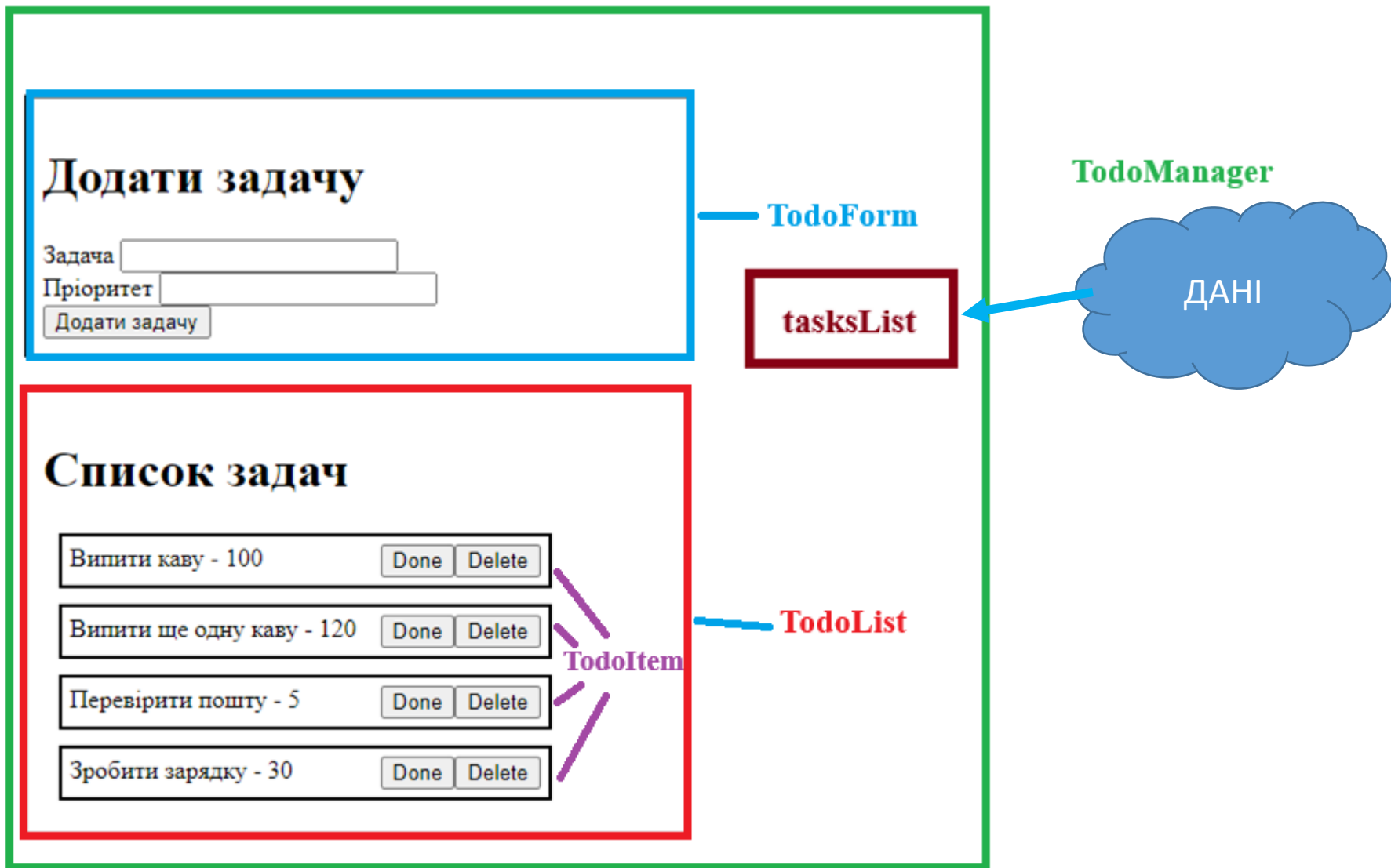
Випити каву - 100	<input type="button" value="Done"/>	<input type="button" value="Delete"/>
Випити ще одну каву - 120	<input type="button" value="Done"/>	<input type="button" value="Delete"/>
Перевірити пошту - 5	<input type="button" value="Done"/>	<input type="button" value="Delete"/>
Зробити зарядку - 30	<input type="button" value="Done"/>	<input type="button" value="Delete"/>

TodoItem

— **TodoList**

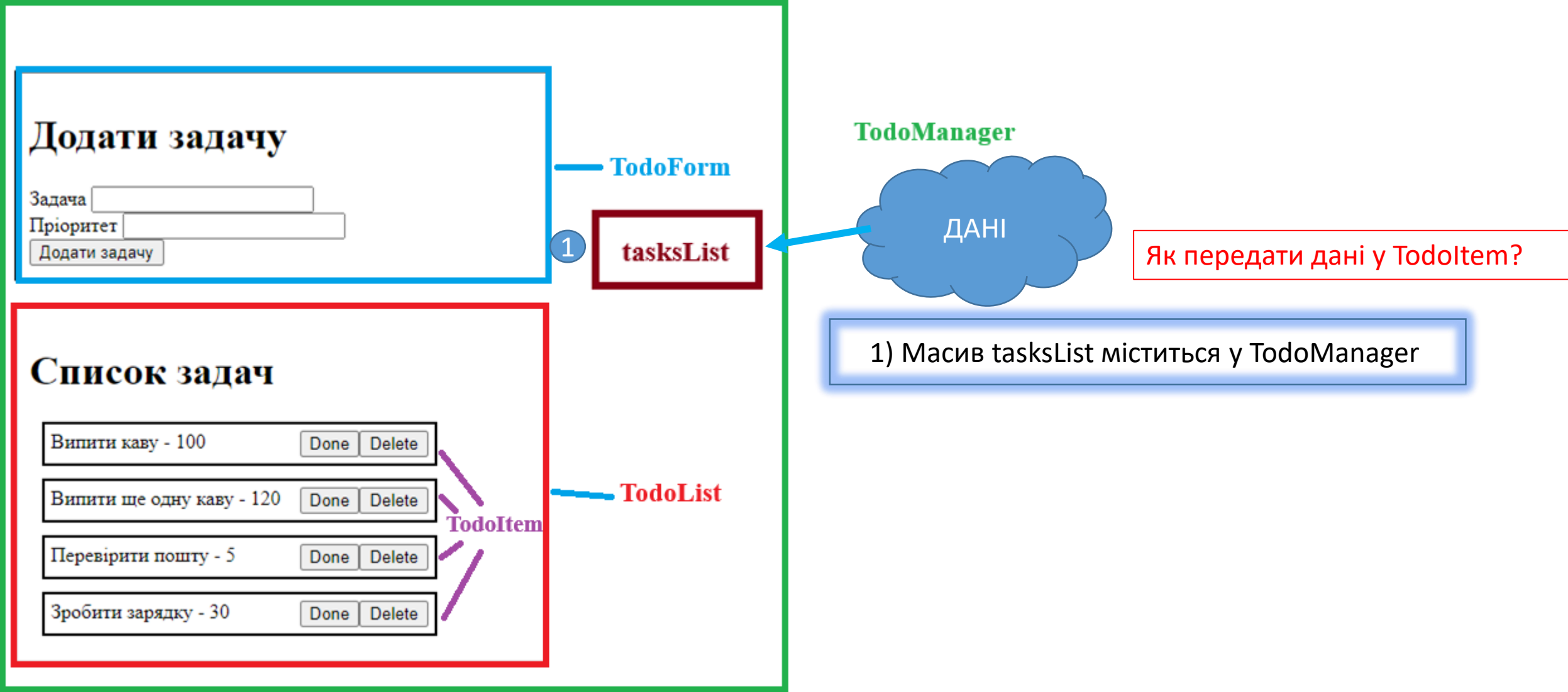
Для чого потрібе Vuex?

Розглянемо меджер задач



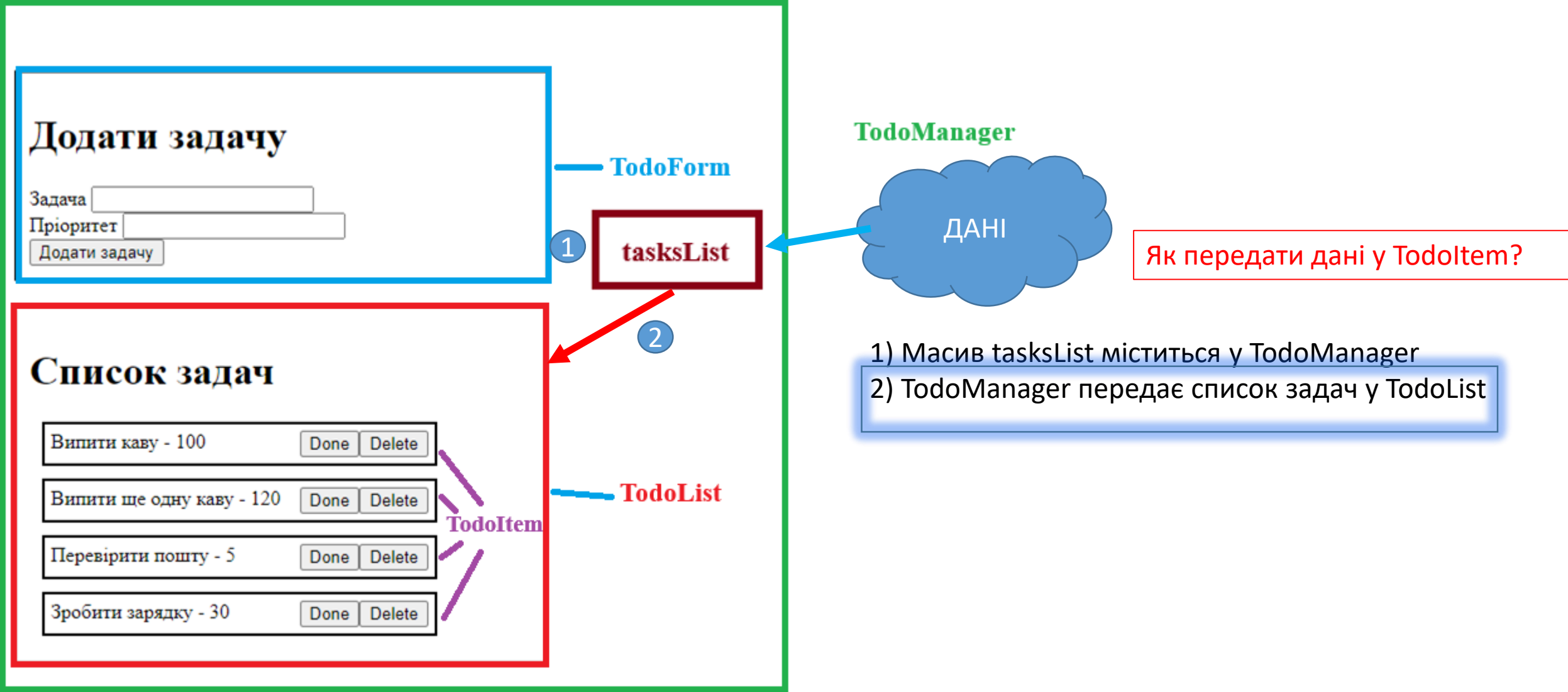
Для чого потрібе Vuex?

Розглянемо меджер задач



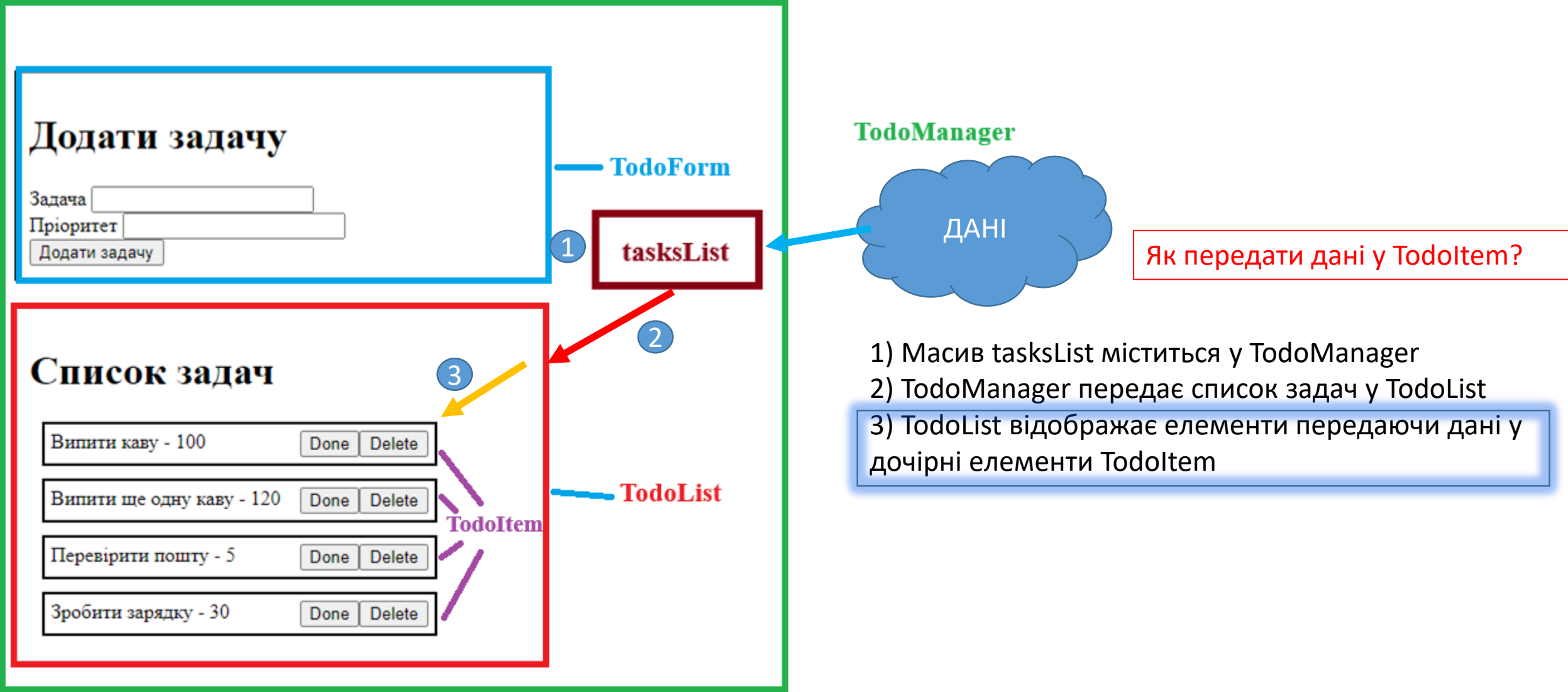
Для чого потрібе Vuex?

Розглянемо меджер задач



Для чого потрібе Vuex?

Розглянемо меджер задач



```

TodoManager.vue components U X
components > TodoManager.vue > {} "TodoManager.vue" > scrip
1 <template>
2   <todo-form @add="addTask" />
3   <todo-list :tasks-list="todoListData"
4     @done="onDone" @delete="onDelete" />
5 </template>
6
7 <script>
8 import TodoForm from './TodoForm.vue'
9 import TodoList from './TodoList.vue'
10 export default {
11   name: 'TodoManager',
12   components: { TodoForm, TodoList },
13
14   data() {
15     return {
16       todoListData: [],
17     },
18   },
19
20   methods: { ...
34   },
35 }
36 </script>
37 TodoForm
38
39 <style lang="scss" scoped></style>
40

```

```

TodoList.vue components U
src > components > TodoList.vue > {} "TodoList.vue" > template
1 <template>
2   <div class="list-container">
3     <h1>Список задач</h1>
4     <todo-item
5       v-for="task in tasksList"
6       :key="task.id"
7       :task="task"
8       @done="$emit('done', $event)"
9       @delete="$emit('delete', $event)"
10     />
11   </div>
12 </template>
13 <script>
14 import TodoItem from './TodoItem.vue'
15 export default {
16   name: 'TodoList',
17
18   components: { TodoItem },
19   emits: ['done', 'delete'],
20   props: {
21     tasksList: {
22       type: Array,
23       default: () => [],
24     },
25   },
26 }
27 </script>

```

```

TodoItem.vue components U X
src > components > TodoItem.vue > {} "TodoItem.vue" > script
1 <template>
2   <div class="task-container">
3     <div>{{ task.title }} - {{ task.priority }}</div>
4     <div>
5       <button v-if="!isDone" @click="$emit('done', task
6         id)">Done</button>
7       <button @click="$emit('delete', task.id)">Delete<
8         button>
9     </div>
10   </div>
11 </template>
12 <script>
13 export default {
14   name: 'CustomInput',
15
16   props: {
17     task: {
18       type: Object,
19       required: true,
20     },
21   },
22   computed: { ...
26   },
27 }
28 </script>

```

1) Масив tasksList міститься у TodoManager


```

components > TodoManager.vue components U
1 <template>
2   <todo-form @add="addTask" />
3   <todo-list :tasks-list="todoListData"
4     @done="onDone" @delete="onDelete" />
5 </template>
6
7 <script>
8 import TodoForm from './TodoForm.vue'
9 import TodoList from './TodoList.vue'
10 export default {
11   name: 'TodoManager',
12   components: { TodoForm, TodoList },
13
14   data() {
15     return {
16       todoListData: [],
17     }
18   },
19
20   methods: { ...
34   },
35 }
36 </script>
37 TodoForm
38
39 <style lang="scss" scoped></style>
40

```

```

src > components > TodoList.vue components U
1 <template>
2   <div class="list-container">
3     <h1>Список задач</h1>
4     <todo-item
5       v-for="task in tasksList"
6       :key="task.id"
7       :task="task"
8       @done="$emit('done', $event)"
9       @delete="$emit('delete', $event)"
10    />
11   </div>
12 </template>
13 <script>
14 import TodoItem from './TodoItem.vue'
15 export default {
16   name: 'TodoList',
17
18   components: { TodoItem },
19   emits: ['done', 'delete'],
20   props: {
21     tasksList: {
22       type: Array,
23       default: () => [],
24     },
25   },
26 }
27 </script>

```

```

src > components > TodoItem.vue components U
1 <template>
2   <div class="task-container">
3     <div>{{ task.title }} - {{ task.priority }}</div>
4     <div>
5       <button v-if="!isDone" @click="$emit('done', task
6         id)">Done</button>
7       <button @click="$emit('delete', task.id)">Delete<
8         button>
9     </div>
10   </div>
11 </template>
12 <script>
13 export default {
14   name: 'CustomInput',
15
16   props: {
17     task: {
18       type: Object,
19       required: true,
20     },
21   },
22   computed: { ...
26   },
27 }
28 </script>

```

1) Масив tasksList міститься у TodoManager

2) TodoManager передає список задач у TodoList

components > ▼ TodoManager.vue components U ×

```

1 <template>
2   <todo-form @add="addTask" />
3   <todo-list :tasks-list="todoListData"
4     @done="onDone" @delete="onDelete" />
5 </template>
6
7 <script>
8 import TodoForm from './TodoForm.vue'
9 import TodoList from './TodoList.vue'
10 export default {
11   name: 'TodoManager',
12   components: { TodoForm, TodoList },
13
14   data() {
15     return {
16       todoListData: [],
17     }
18   },
19
20   methods: { ...
34   },
35 }
36 </script>
37 TodoForm
38
39 <style lang="scss" scoped></style>
40

```

1

2

src > components > ▼ TodoList.vue components U ●

```

1 <template>
2   <div class="list-container">
3     <h1>Список задач</h1>
4     <todo-item
5       v-for="task in tasksList"
6       :key="task.id"
7       :task="task"
8       @done="$emit('done', $event)"
9       @delete="$emit('delete', $event)"
10     />
11   </div>
12 </template>
13 <script>
14 import TodoItem from './TodoItem.vue'
15 export default {
16   name: 'TodoList',
17
18   components: { TodoItem },
19   emits: ['done', 'delete'],
20   props: {
21     tasksList: {
22       type: Array,
23       default: () => [],
24     },
25   },
26 }
27 </script>

```

3

src > components > ▼ TodoItem.vue components U ×

```

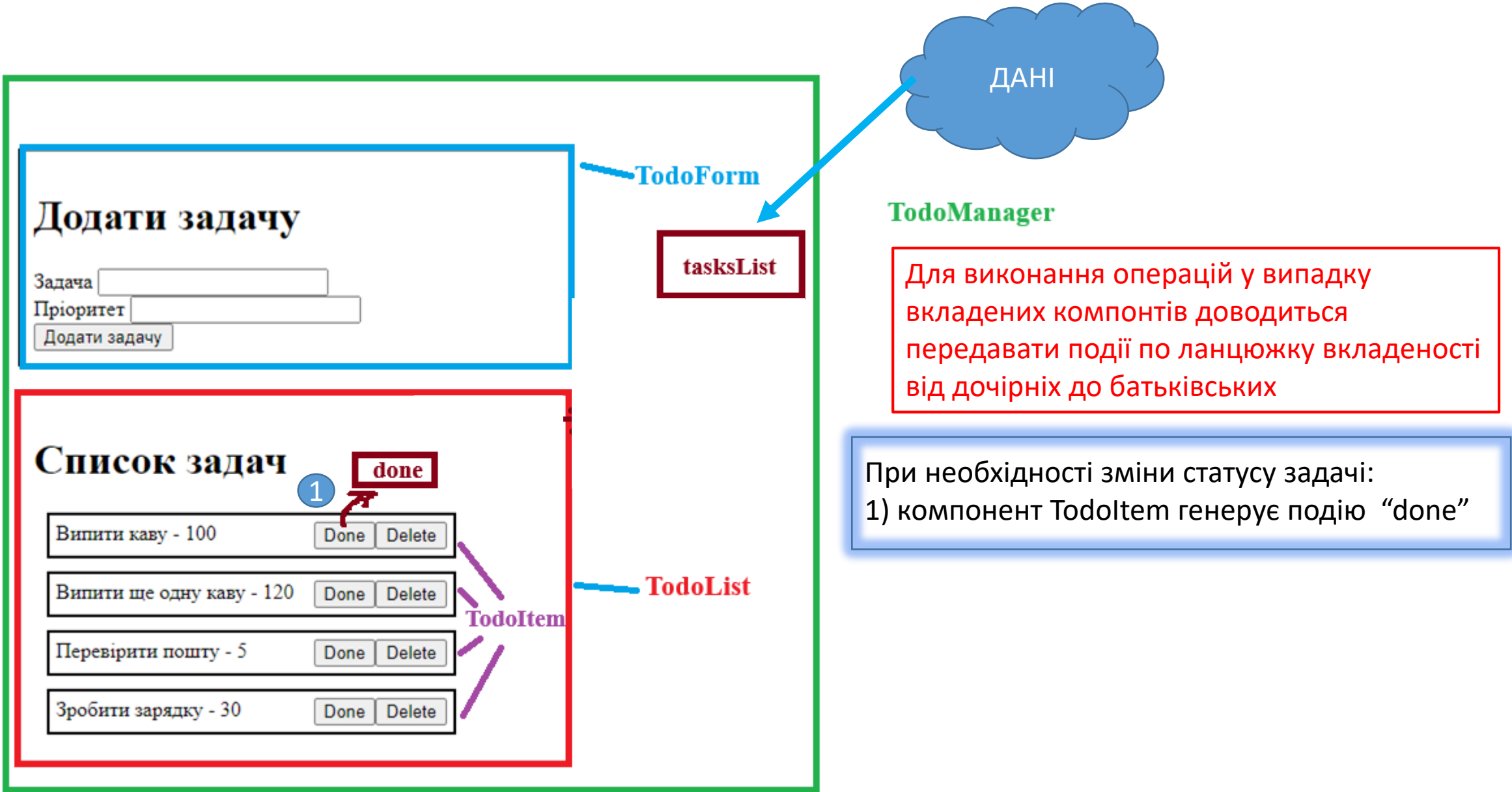
1 <template>
2   <div class="task-container">
3     <div>{{ task.title }} - {{ task.priority }}</div>
4     <div>
5       <button v-if="!isDone" @click="$emit('done', task
6         id)">Done</button>
7       <button @click="$emit('delete', task.id)">Delete<
8         button>
9     </div>
10   </div>
11 </template>
12 <script>
13 export default {
14   name: 'CustomInput',
15
16   props: {
17     task: {
18       type: Object,
19       required: true,
20     },
21   },
22
23   computed: { ...
26   },
27 }
28 </script>

```

- 1) Масив tasksList міститься у TodoManager
- 2) TodoManager передає список задач у TodoList
- 3) TodoList відображає елементи

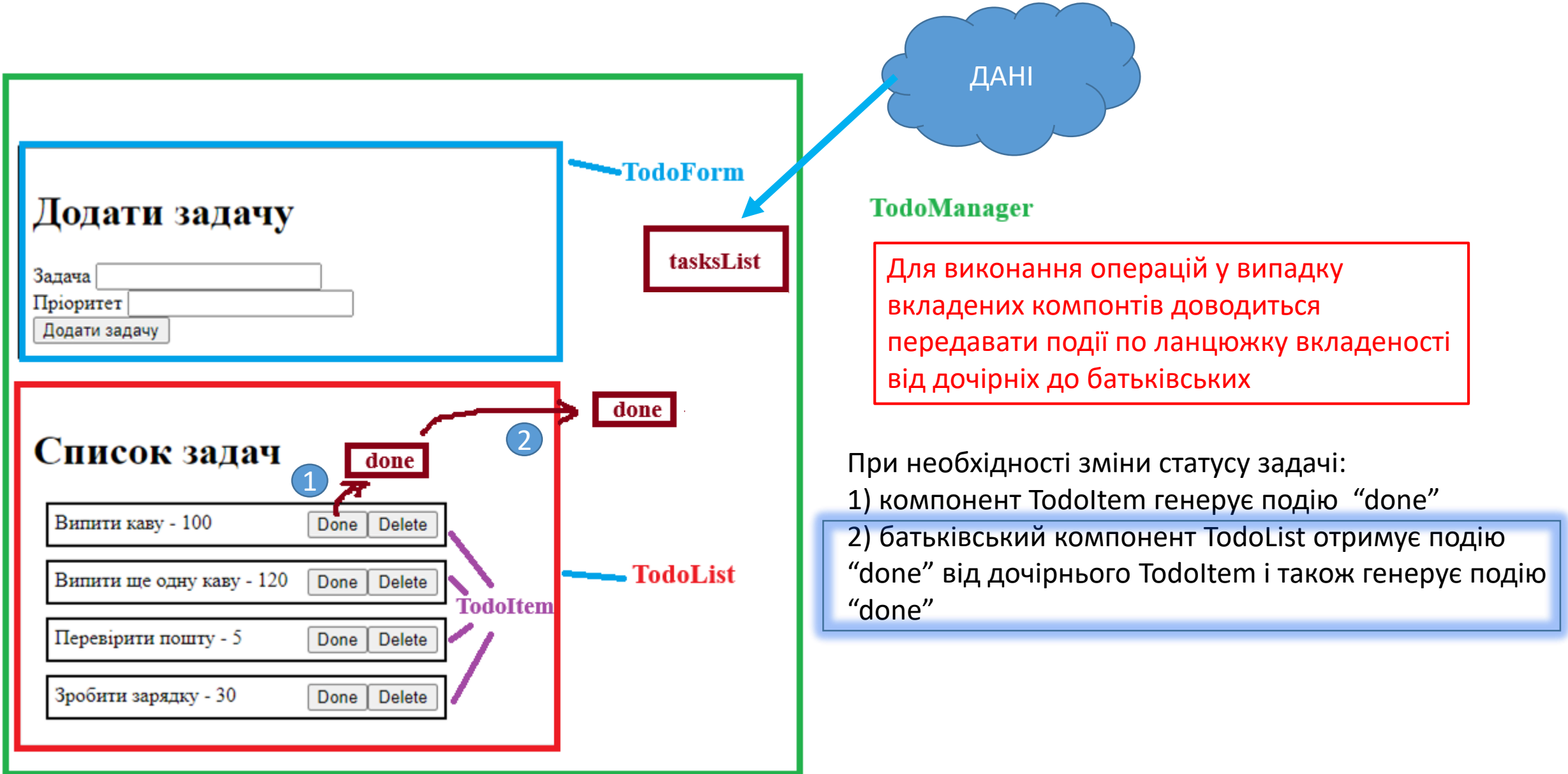
Для чого потрібе Vuex?

Розглянемо меджер задач



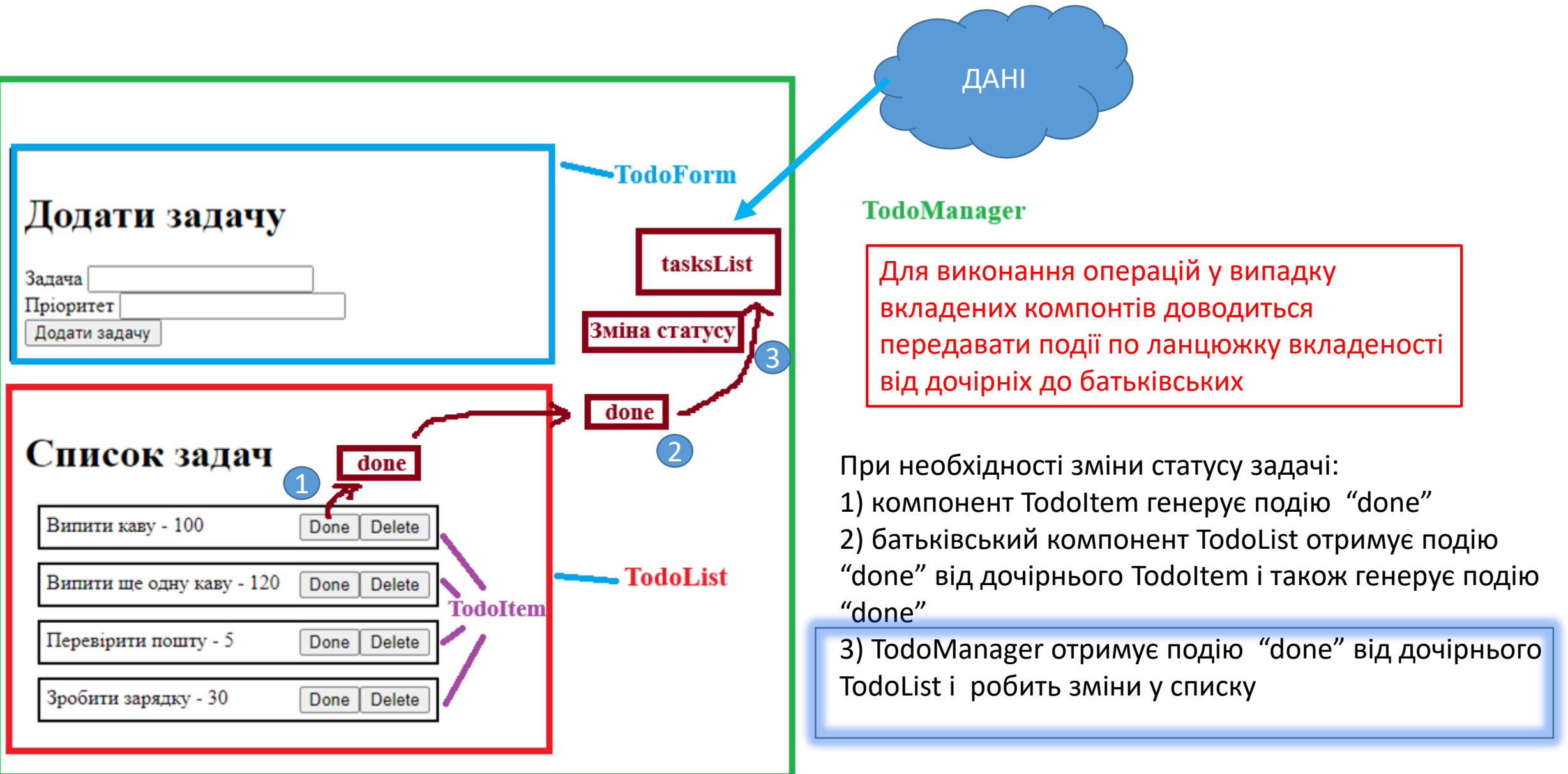
Для чого потрібе Vuex?

Розглянемо меджер задач



Для чого потрібе Vuex?

Розглянемо меджер задач



Для чого потрібе Vuex?

Розглянемо меджер задач

```
▼ TodoManager.vue components U ×
> ▼ TodoManager.vue > {} "TodoManager.vue" > script > default
1 <template>
2   <todo-form @add="addTask" />
3   <todo-list :tasks-list="todoListData"
4     @done="onDone" @delete="onDelete" />
5 </template>
6 <script>
7 import TodoForm from './TodoForm.vue'
8 import TodoList from './TodoList.vue'
9 export default {
10   name: 'TodoManager',
11   components: { TodoForm, TodoList },
12   data() {
13     return {
14       todoListData: [],
15     },
16   },
17   methods: { ...
18   },
19   },
20   },
21   },
22   },
23   },
24   },
25   },
26   },
27   },
28   },
29   },
30   },
31   },
32   },
33   },
34   },
35   },
36   },
37   },
38   },
39   }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
```

```
▼ TodoList.vue components U ●
src > components > ▼ TodoList.vue > {} "TodoList.vue" > script > default
1 <template>
2   <div class="list-container">
3     <h1>Список задач</h1>
4     <todo-item
5       v-for="task in tasksList"
6       :key="task.id"
7       :task="task"
8       @done="$emit('done', $event)"
9       @delete="$emit('delete', $event)"
10     />
11   </div>
12 </template>
13 <script>
14 import TodoItem from './TodoItem.vue'
15 export default {
16   name: 'TodoList',
17   components: { TodoItem },
18   emits: ['done', 'delete'],
19   props: {
20     tasksList: {
21       type: Array,
22       default: () => [],
23     },
24   },
25 },
26 }
27 </script>
```

```
▼ TodoItem.vue components U ●
src > components > ▼ TodoItem.vue > {} "TodoItem.vue"
1 <template>
2   <div class="task-container">
3     <div {{ task.title }} - {{ task.priority }}</div>
4     <div>
5       <button v-if="!isDone" @click="$emit('done', task.id)">
6         Done</button>
7       <button @click="$emit('delete', task.id)">Delete</button>
8     </div>
9   </template>
10 <script>
11 export default {
12   name: 'CustomInput',
13   props: {
14     task: {
15       type: Object,
16       required: true,
17     },
18   },
19   computed: { ...
20   },
21   },
22   },
23   },
24   },
25   },
26   },
27   },
28   },
29   },
30   },
31   },
32   },
33   },
34   },
35   },
36   },
37   },
38   },
39   },
40   },
41   },
42   },
43   },
44   },
45   },
46   },
47   },
48   },
49   },
50   },
51   },
52   },
53   },
54   },
55   },
56   },
57   },
58   },
59   },
60   },
61   },
62   },
63   },
64   },
65   },
66   },
67   },
68   },
69   },
70   },
71   },
72   },
73   },
74   },
75   },
76   },
77   },
78   },
79   },
80   },
81   },
82   },
83   },
84   },
85   },
86   },
87   },
88   },
89   },
90   },
91   },
92   },
93   },
94   },
95   },
96   },
97   },
98   },
99   },
100 }
```

При необхідності зміни статусу задачі:

1) компонент TodoItem генерує подію "done"

Для чого потрібе Vuex?

Розглянемо меджер задач

```
TodoManager.vue components U X
1 <template>
2   <todo-form @add="addTask" />
3   <todo-list :tasks-list="todoListData"
4     @done="onDone" @delete="onDelete" />
5 </template>
6 <script>
7   import TodoForm from './TodoForm.vue'
8   import TodoList from './TodoList.vue'
9   export default {
10     name: 'TodoManager',
11     components: { TodoForm, TodoList },
12     data() {
13       return {
14         todoListData: [],
15       }
16     },
17     methods: { ...
34   },
35 }
36 </script>
37 TodoForm
38
39 <style lang="scss" scoped></style>

TodoList.vue components U
src > components > TodoList.vue > {} "TodoList.vue" > script >
1 <template>
2   <div class="list-container">
3     <h1>Список задач</h1>
4     <todo-item
5       v-for="task in tasksList"
6       :key="task.id"
7       :task="task"
8       @done="$emit('done', $event)"
9       @delete="$emit('delete', $event)"
10    />
11   </div>
12 </template>
13 <script>
14   import TodoItem from './TodoItem.vue'
15   export default {
16     name: 'TodoList',
17     components: { TodoItem },
18     emits: ['done', 'delete'],
19     props: {
20       tasksList: {
21         type: Array,
22         default: () => [],
23       },
24     },
25   },
26 }
27 </script>

TodoItem.vue components U
src > components > TodoItem.vue > {} "TodoItem.vue"
1 <template>
2   <div class="task-container">
3     <div>{{ task.title }} - {{ task.priority }}</div>
4     <div>
5       <button v-if="!isDone" @click="$emit('done', task.id)">
6         Done</button>
7       <button @click="$emit('delete', task.id)">Delete</button>
8     </div>
9   </template>
10
11 <script>
12   export default {
13     name: 'CustomInput',
14     props: {
15       task: {
16         type: Object,
17         required: true,
18       },
19     },
20   },
21   computed: { ...
26   },
27 }
28 </script>
29
```

При необхідності зміни статусу задачі:

- 1) компонент TodoItem генерує подію "done"
- 2) батьківський компонент TodoList отримує подію "done" від дочірнього TodoItem і також генерує подію "done"

Для чого потрібе Vuex?

Розглянемо меджер задач

```
TodoManager.vue components U X
1 <template>
2   <todo-form @add="addTask" />
3   <todo-list :tasks-list="todoListData"
4     @done="onDone" @delete="onDelete" />
5 </template>
6 <script>
7   import TodoForm from './TodoForm.vue'
8   import TodoList from './TodoList.vue'
9   export default {
10     name: 'TodoManager',
11     components: { TodoForm, TodoList },
12     data() {
13       return {
14         todoListData: [],
15       }
16     },
17     methods: { ...
34   },
35 }
36 </script>
37 TodoForm
38
39 <style lang="scss" scoped></style>

TodoList.vue components U
src > components > TodoList.vue > {} "TodoList.vue" > script >
1 <template>
2   <div class="list-container">
3     <h1>Список задач</h1>
4     <todo-item
5       v-for="task in tasksList"
6       :key="task.id"
7       :task="task"
8       @done="$emit('done', $event)"
9       @delete="$emit('delete', $event)"
10    />
11   </div>
12 </template>
13 <script>
14   import TodoItem from './TodoItem.vue'
15   export default {
16     name: 'TodoList',
17     components: { TodoItem },
18     emits: ['done', 'delete'],
19     props: {
20       tasksList: {
21         type: Array,
22         default: () => [],
23       },
24     },
25   },
26 }
27 </script>

TodoItem.vue components U
src > components > TodoItem.vue > {} "TodoItem.vue"
1 <template>
2   <div class="task-container">
3     <div>{{ task.title }} - {{ task.priority }}</div>
4     <div>
5       <button v-if="!isDone" @click="$emit('done', task.id)">
6         Done</button>
7       <button @click="$emit('delete', task.id)">Delete</button>
8     </div>
9   </template>
10
11 <script>
12   export default {
13     name: 'CustomInput',
14     props: {
15       task: {
16         type: Object,
17         required: true,
18       },
19     },
20   },
21
22   <computed> { ...
26   },
27 }
28 </script>
29
```

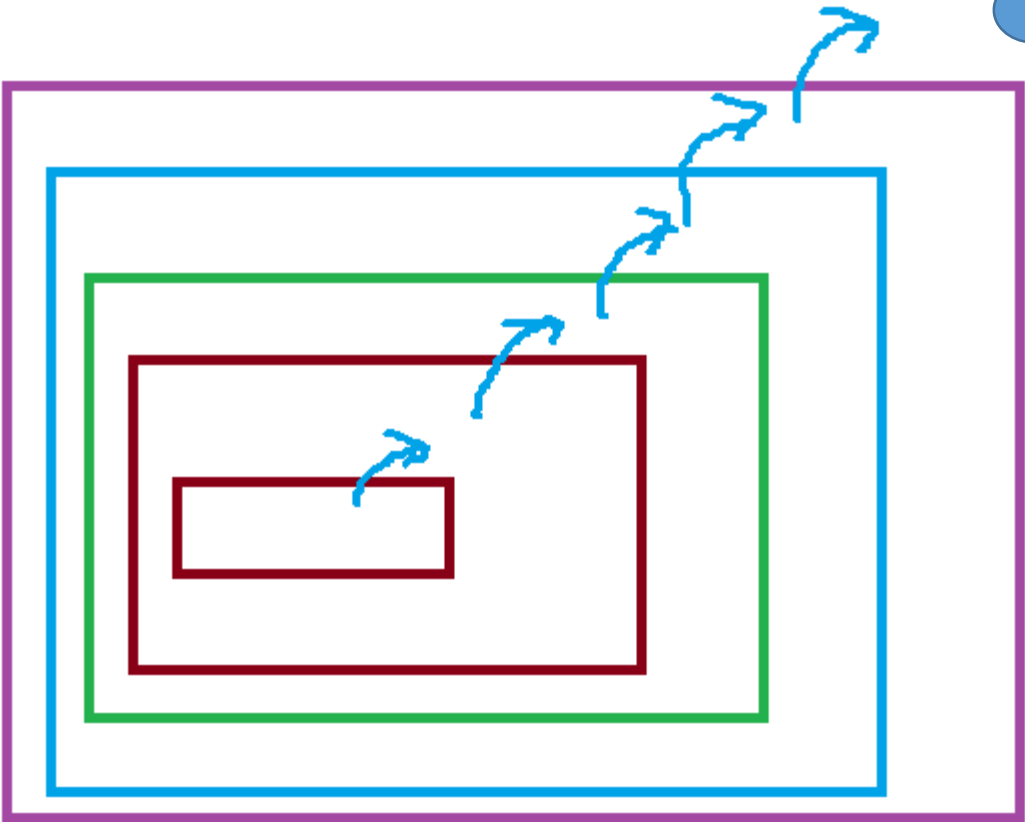
При необхідності зміни статусу задачі:

- 1) компонент TodoItem генерує подію "done"
- 2) батьківський компонент TodoList отримує подію "done" від дочірнього TodoItem і також генерує подію "done"
- 3) TodoManager отримує подію "done" від дочірнього TodoList і робить зміни у списку

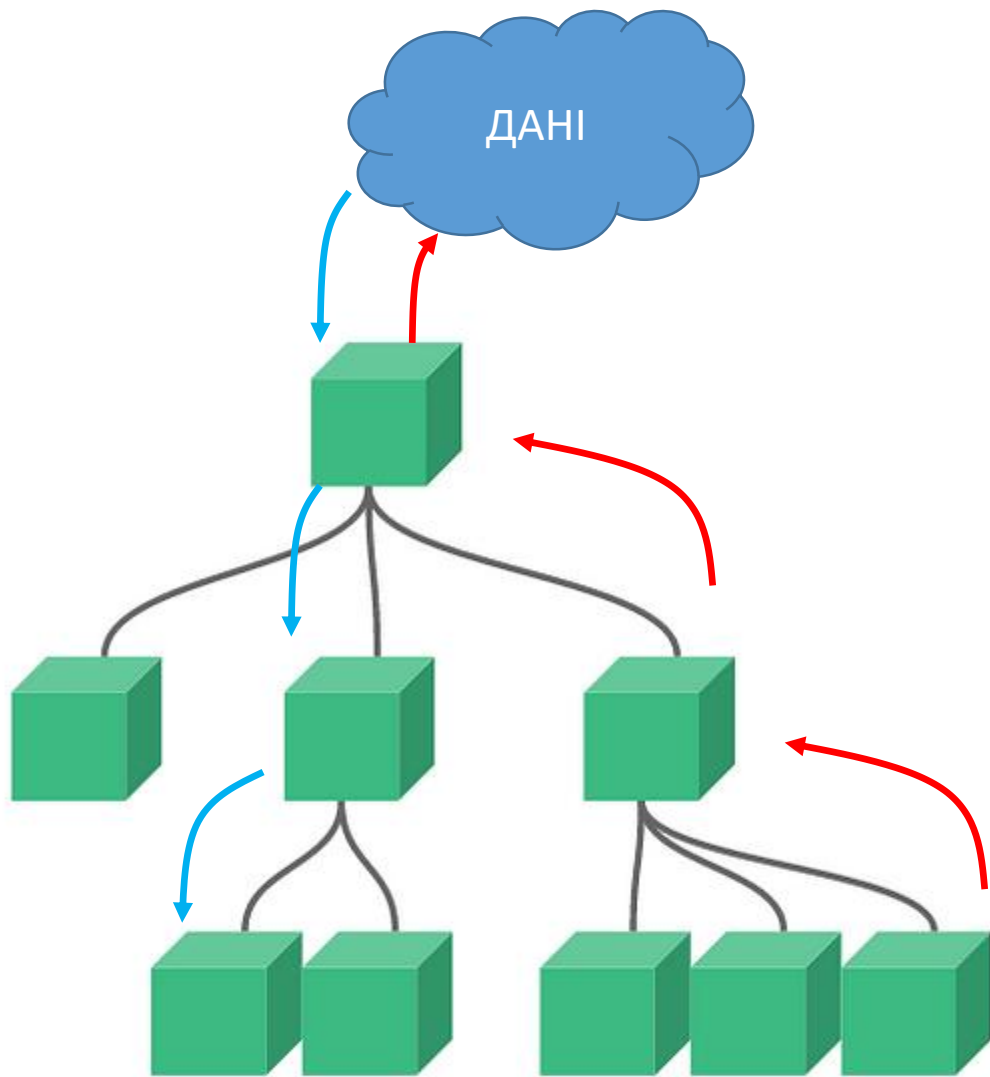
Для чого потрібе Vuex?



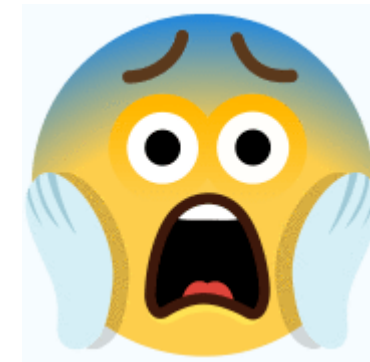
А що буде, якщо глибина
буде ще більшою?



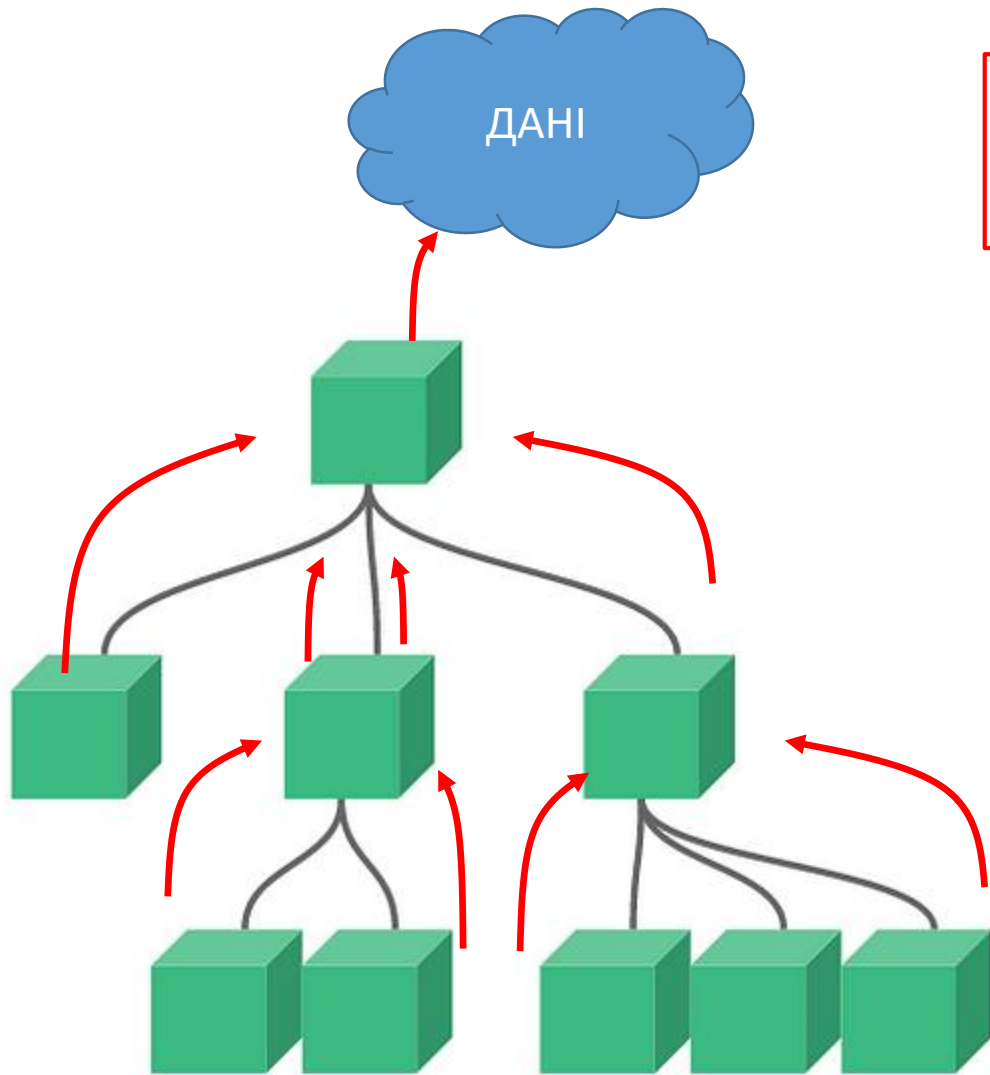
Для чого потрібе Vuex?



Що робити, якщо треба передати дані від одного вкладеного компонента до іншого?



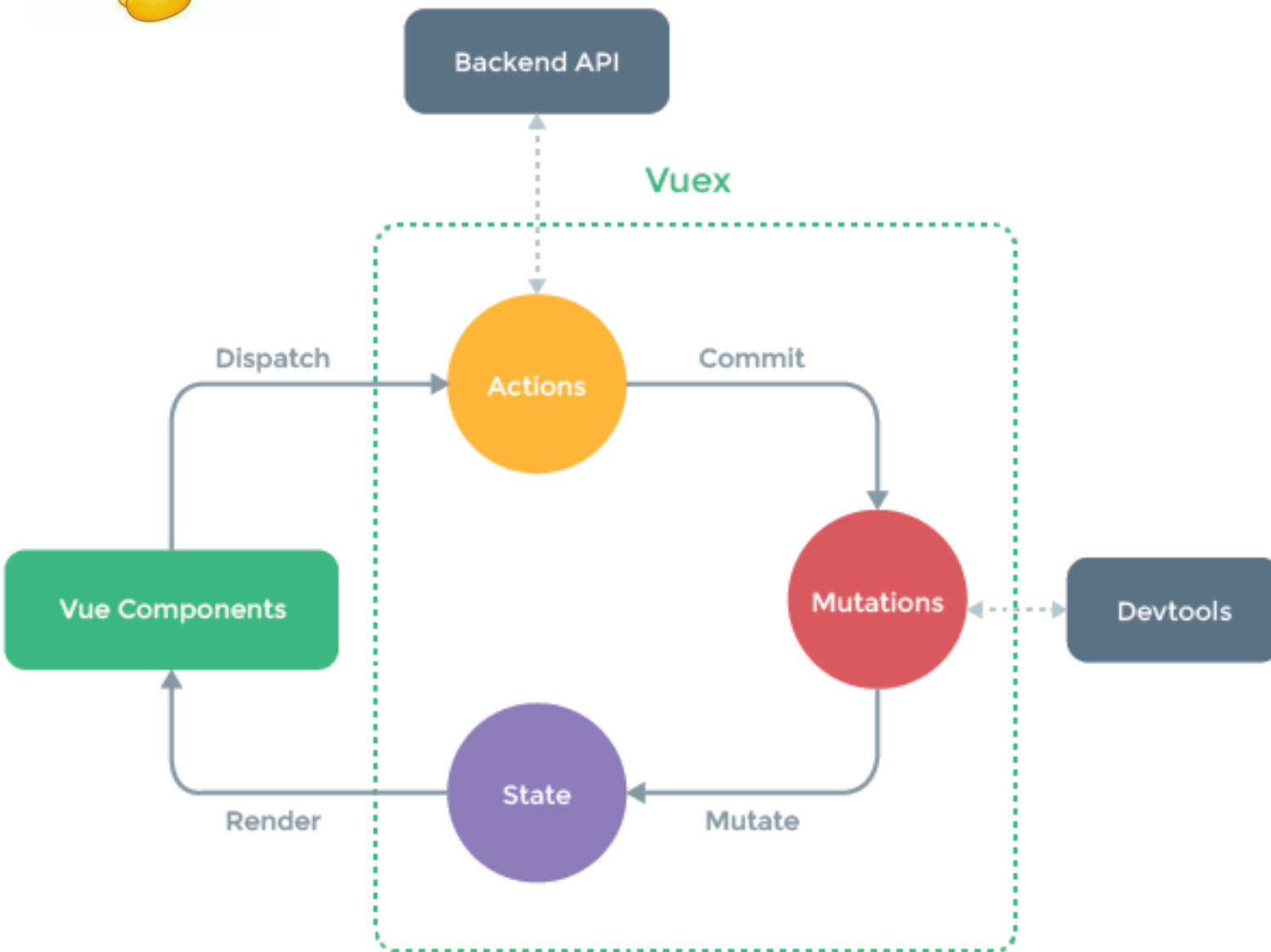
Для чого потрібе Vuex?



А що буде, якщо дані можуть змінюватись різними компонентами? Як це відслідковувати і контролювати?



Для чого потрібе Vuex?

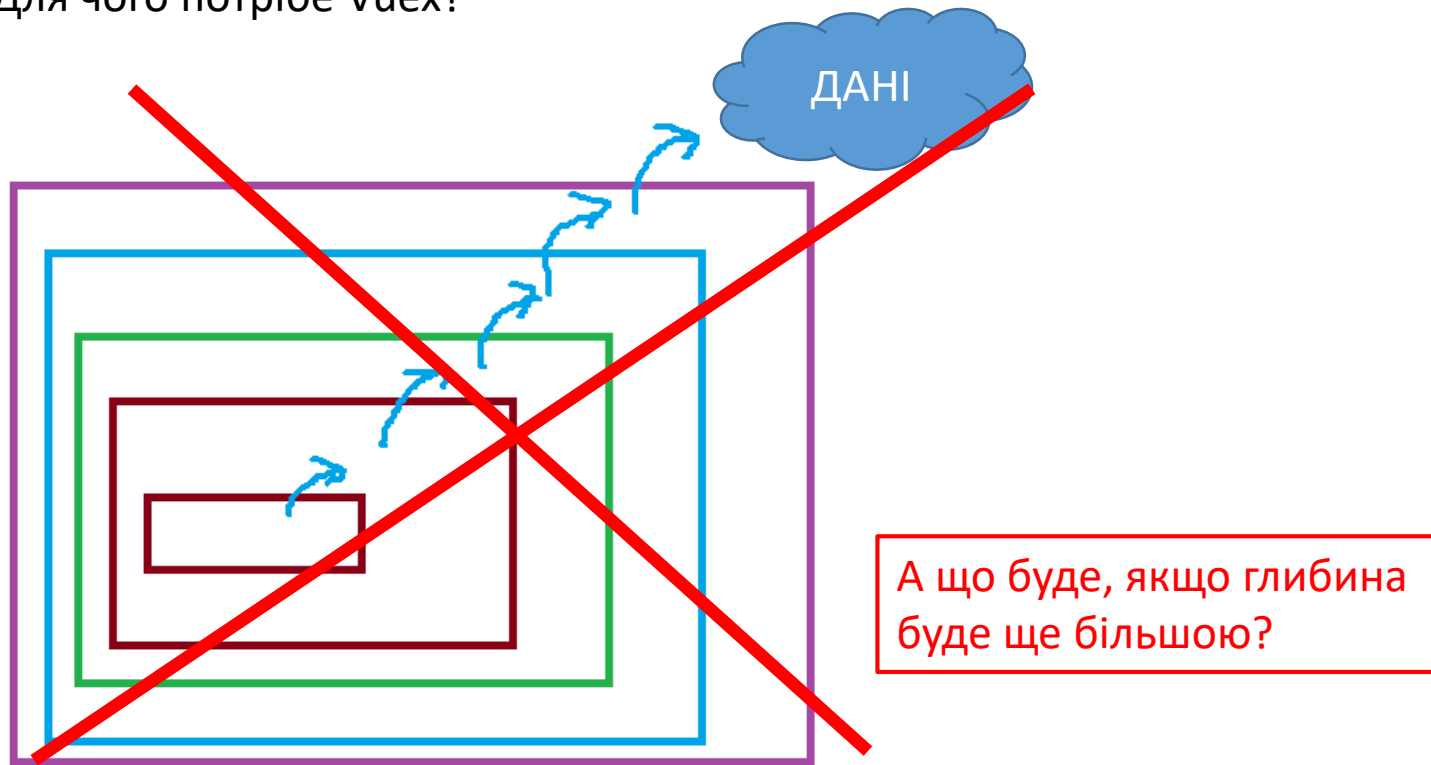


Flux-архитектура – набір шаблонів програмування для побудови інтерфейсу веб-додатків заснований на односторонніх потоках даних

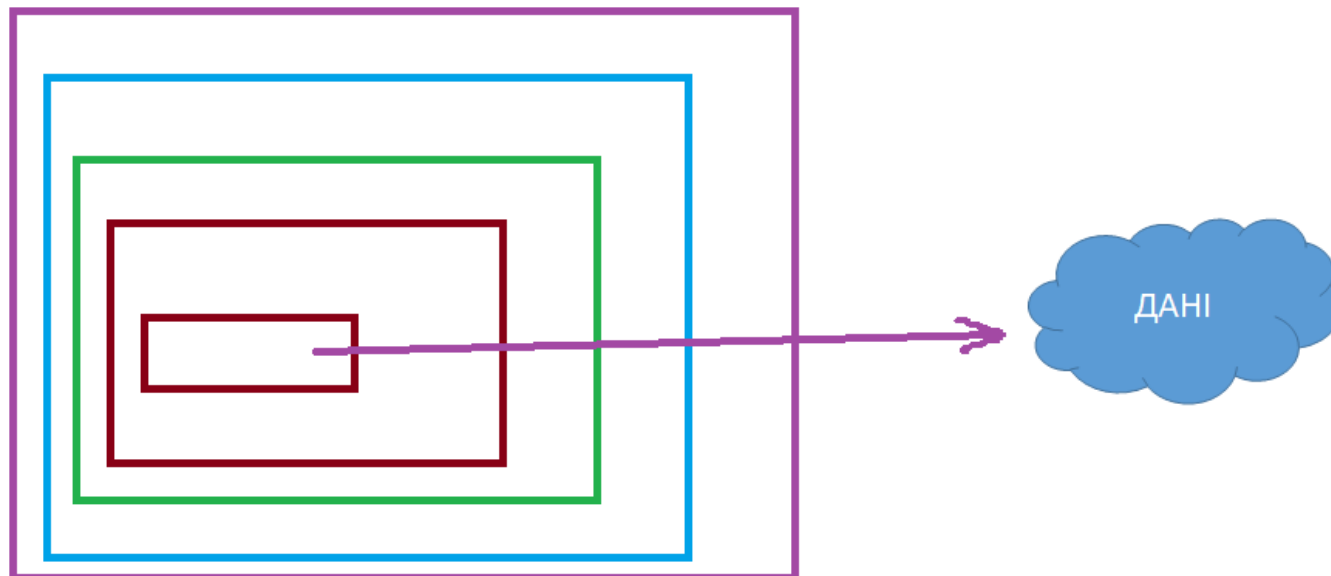
Принципи :

1. Існує єдине глобальне сховище **State**
2. Всі компоненти можуть читати значення безпосередньо з сховища **State**
3. Компоненти не можуть безпосередньо змінювати значення у сховищі
4. Для зміни:
 - 1) будь-який компонент формує заявку (**dispatch**) для змін, тобто ініціює **action**
 - 2) **action** ініціює виконання змін у сховищі (**commit**), тобто **mutation**
 - 3) **mutation** виконують зміну у сховищі
5. Компоненти отримують оновлені дані і оновлюються за потреби

Для чого потрібе Vuex?



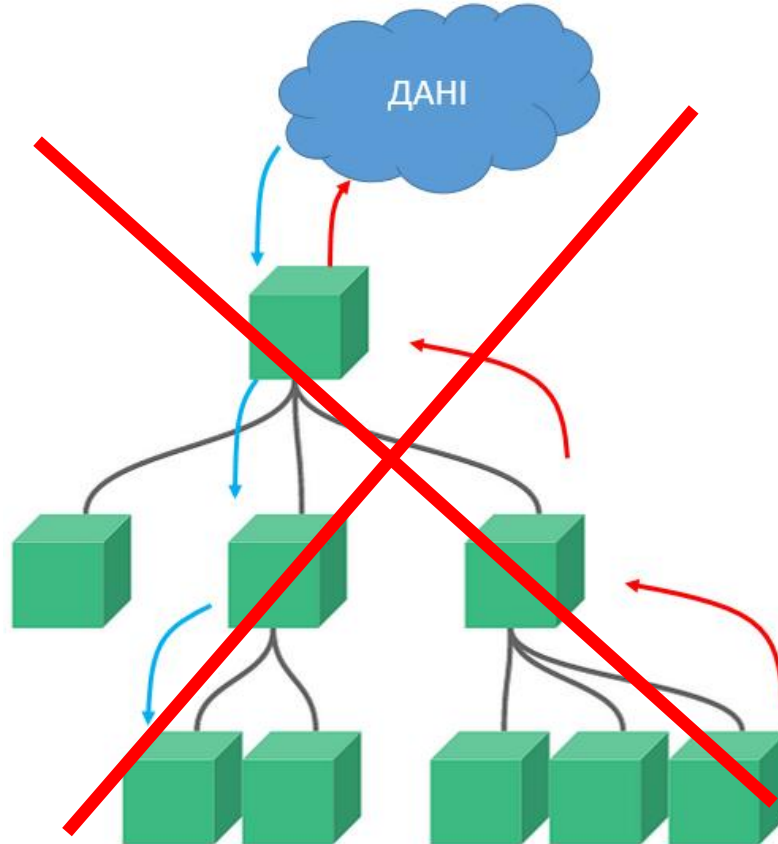
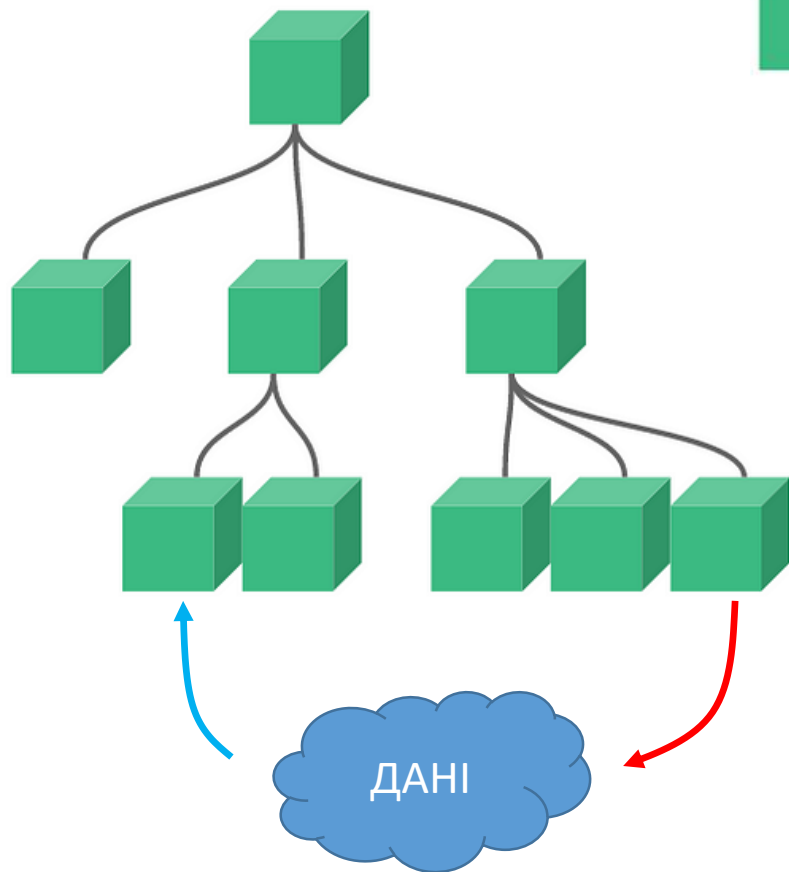
А що буде, якщо глибина
буде ще більшою?



Принципи :

1. Існує єдине глобальне сховище **State**
2. Всі компоненти можуть читати значення безпосередньо з сховища **State**
3. Компоненти не можуть безпосередньо змінювати значення у сховищі
4. Для зміни:
 - 1) будь-який компонент формує заявку для змін, тобто ініціює **action**
 - 2) **action** ініціює виконання змін у сховищі, тобто **mutation**
 - 3) **mutation** виконують зміну у сховищі
5. Компоненти отримують оновлені дані

Що робити, якщо треба
передати дані від одного
вкладеного компонента до
іншого?

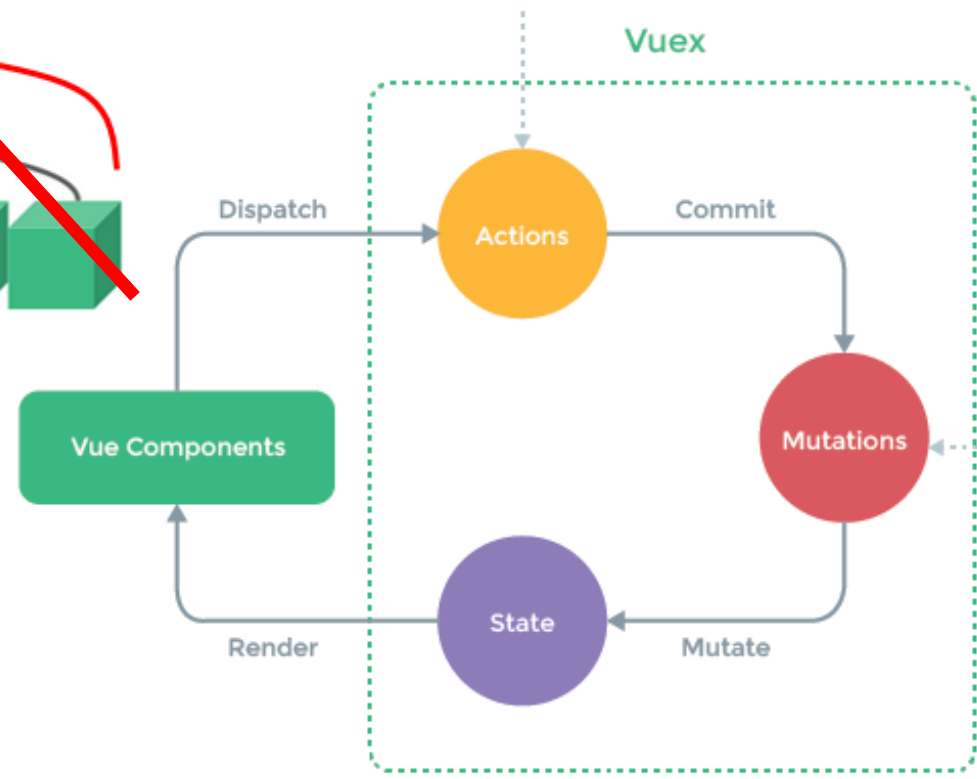
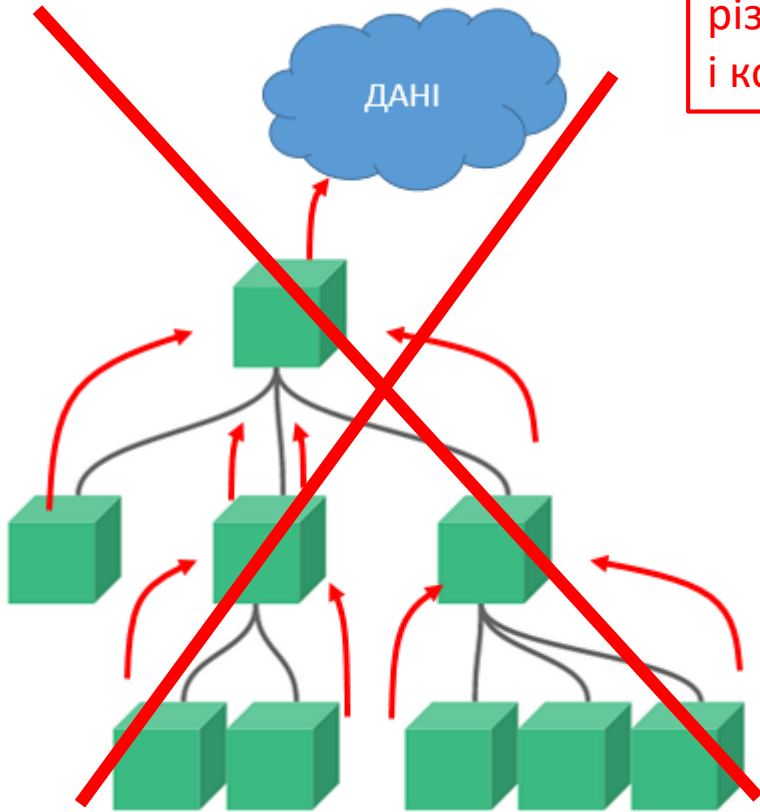


Принципи :

1. Існує єдине глобальне сховище **State**
2. Всі компоненти можуть читати значення безпосередньо з сховища **State**
3. Компоненти не можуть безпосередньо змінювати значення у сховищі
4. Для зміни:
 - 1) будь-який компонент формує заявку для змін, тобто ініціює **action**
 - 2) **action** ініціює виконання змін у сховищі, тобто **mutation**
 - 3) **mutation** виконують зміну у сховищі
5. Компоненти отримують оновлені дані

Для чого потрібе Vuex?

А що буде, якщо дані можуть змінюватись різними компонентами? Як це відслідковувати і контролювати?

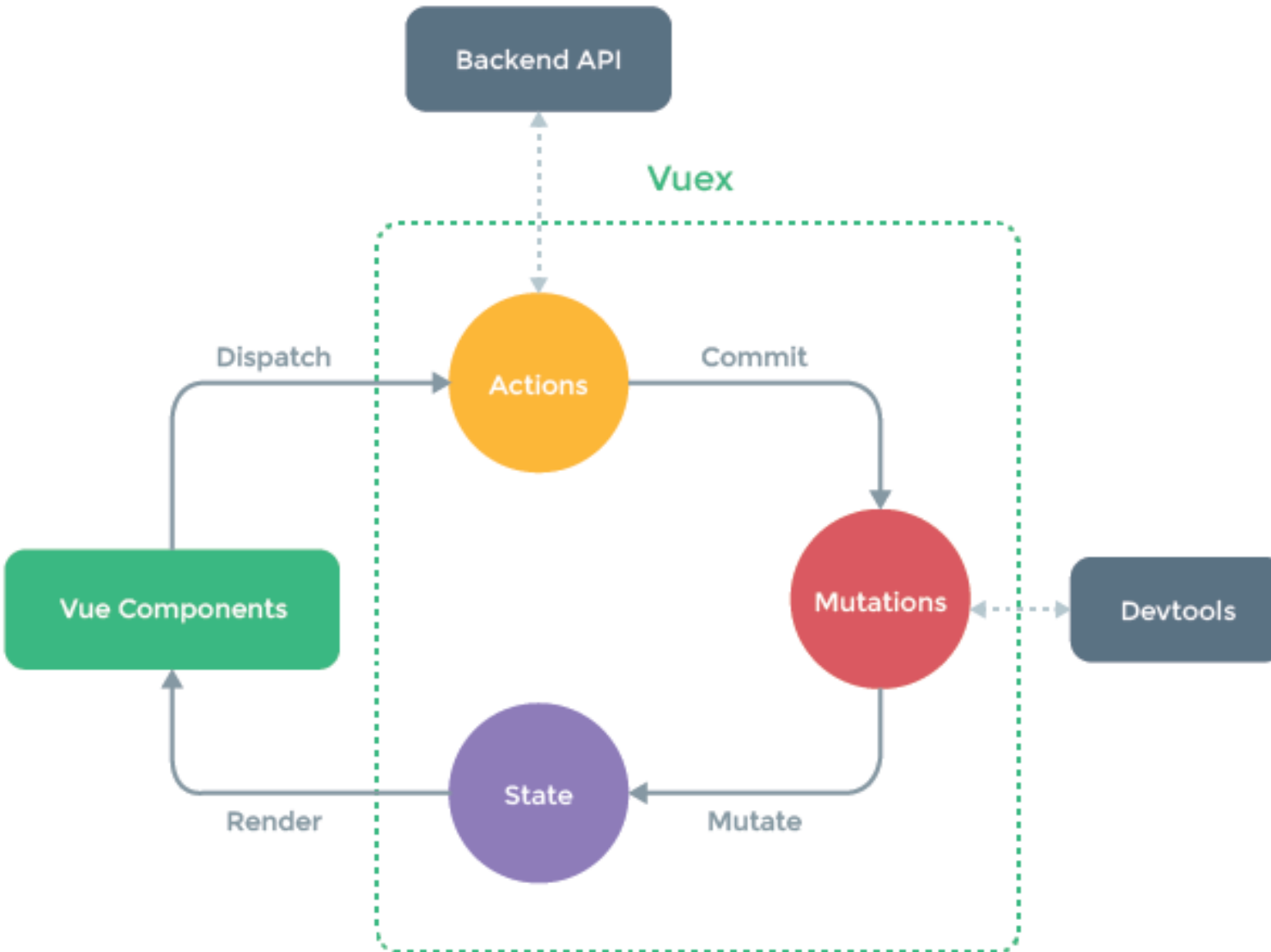


Принципи :

1. Існує єдине глобальне сховище **State**
2. Всі компоненти можуть читати значення безпосередньо з сховища **State**
3. Компоненти не можуть безпосередньо змінювати значення у сховищі
4. Для зміни:
 - 1) будь-який компонент формує заявку для змін, тобто ініціює **action**
 - 2) **action** ініціює виконання змін у сховищі, тобто **mutation**
 - 3) **mutation** виконують зміну у сховищі
5. Компоненти отримують оновлені дані

Для чого потрібе Vuex?

Flux-архітектура – набір шаблонів програмування для побудови інтерфейсу веб-додатків заснований на односторонніх потоках даних



Принципи :

1. Існує єдине глобальне сховище **State**
2. Всі компоненти можуть читати значення безпосередньо з сховища **State**
3. Компоненти не можуть безпосередньо змінювати значення у сховищі
4. Для зміни:
 - 1) компонент формує заявку для змін, тобто ініціює **action**
 - 2) **action** ініціює виконання змін у сховищі, тобто **mutation**
 - 3) **mutation** виконують зміну у сховищі
5. Компоненти отримують оновлені дані


```
import { createStore } from 'vuex'
```

```
// Create a new store instance.  
const store = createStore({
```

```
})
```

```
import { createStore } from 'vuex'
```

```
// Create a new store instance.  
const store = createStore({
```

```
})
```

----- Опис сховища -----
1. Створюємо об'єкт сховища

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  //опис властивостей стану(аналог data)
  state () {
    return {
      властивість1: поч.знач.1,
      властивість2: поч.знач.2,
      . . . . .
    }
  },
})
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
})
```

---- Опис сховища ----

1. Створюємо об'єкт сховища
2. Описуємо властивості сховища, які хочемо відслідковувати

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  //опис властивостей стану(аналог data)
  state () {
    return {
      властивість1: поч.знач.1,
      властивість2: поч.знач.2,
      . . . . .
    }
  },
  //опис обчисл.власт.(аналог computed)
  getters: {
    обч.геттер1(state) {
      обчисл. знач. на основі state
    },
    . . . . .
  },
})
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
    doubleCount(state) {
      return state.count * 2
    },
  },
})
```

----- Опис сховища -----

1. Створюємо об'єкт сховища
2. Описуємо властивості сховища, які хочемо відслідковувати
3. Опис методів-геттерів (аналог обчислювальних властивостей)

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  //опис властивостей стану(аналог data)
  state () {
    return {
      властивість1: поч.знач.1,
      властивість2: поч.знач.2,
      . . . . .
    }
  },
  //опис обчисл.власт.(аналог computed)
  getters: {
    обч.геттер1(state) {
      обчисл. знач. на основі state
    },
    . . . . .
  },
})
```

Повна форма містить 4 параметри
(state, getters, rootState, rootGetters)

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,

    doubleCount(state) {
      return state.count * 2
    },
  },
})
```

- Опис сховища ----
1. Створюємо об'єкт сховища
 2. Описуємо властивості сховища, які хочемо відслідковувати
 3. Опис методів-геттерів (аналог обчислювальних властивостей)

Приклад. коли треба інші параметри

```
getters: {
  // ...
  геттер (state, getters) {
    використовуємо якийсь інший
    getters.інший_геттер
  }
}
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  //опис властивостей стану(аналог data)
  state () {
    return {
      властивість1: поч.знач.1,
      властивість2: поч.знач.2,
      . . . . .
    }
  },
  //опис обчисл.власт.(аналог computed)
  getters: {
    обч.геттер1(state) {
      обчисл. знач. на основі state
    },
    . . . . .
  },
  //опис методів зміни стану
  mutations: {
    метод_мутація1(state) {
      зміна властивостей state
    },
    . . . . .
  },
})
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,

    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
})
```

---- Опис сховища ----

1. Створюємо об'єкт сховища
2. Описуємо властивості сховища, які хочемо відслідковувати
3. Опис методів-геттерів (аналог обчислювальних властивостей)
4. Опис мутацій – методів, які можуть змінювати значення у state

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  //опис властивостей стану(аналог data)
  state () {
    return {
      властивість1: поч.знач.1,
      властивість2: поч.знач.2,
      . . . . .
    }
  },
  //опис обчисл.власт.(аналог computed)
  getters: {
    обч.геттер1(state) {
      обчисл. знач. на основі state
    },
    . . . . .
  },
  //опис методів зміни стану
  mutations: {
    метод_мутація1(state) {
      зміна властивостей state
    },
    . . . . .
  },
  //опис методів
  actions: {
    increment (context) {
      context.commit('increment')
    },
    . . . . .
  }
})
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})
```

---- Опис сховища ----

1. Створюємо об'єкт сховища
2. Описуємо властивості сховища, які хочемо відслідковувати
3. Опис методів-геттерів (аналог обчислювальних властивостей)
4. Опис мутацій – методів, які можуть змінювати значення у state
5. Опис action-методів – методів, які викликаються з компонентів для зміни значення сховища (не безпосередньо, а з використанням мутацій, які викликаються з використанням метода

commit ('назва мутації')

Доступ до сховища у компонентах. Пряме звертання до сховища з використанням *this.\$store*

this . \$store



```
▼ Store i
  ► commit: f boundCommit(type, payload, options)
  ► dispatch: f boundDispatch(type, payload)
  ► getters: {}
  ▼ state: Proxy(Object)
    ► [[Handler]]: MutableReactiveHandler
    ▼ [[Target]]: Object
      count: 0
    ► [[Prototype]]: Object
      [[IsRevoked]]: false
    ► [[Prototype]]: Object
```

Доступ до сховища у компонентах. Пряме звертання до сховища з використанням *this.\$store*

this . \$store



Для виклику мутацій

Для виклику action-методів

Для звертання до
геттера

Для прямого
звертання до
властивостей
state

this.\$store.**state**.count

```
▼ Store ⓘ
  ► commit: f boundCommit(type, payload, options)
  ► dispatch: f boundDispatch(type, payload)
  ► getters: {}
  ▼ state: Proxy(Object)
    ► [[Handler]]: MutableReactiveHandler
    ▼ [[Target]]: Object
      count: 0
    ► [[Prototype]]: Object
      [[IsRevoked]]: false
    ► [[Prototype]]: Object
```

----->

Пряме зчитування **властивостей** з сховища (state)

Загальна форма

this.\$store.state. **властивість**

Фрагмент компонента

```
<template>
  <div>{{ $store.state.count }}</div>
</template>
```

Сховище

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    doubleCount(state) {
      return state.count * 2
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Пряме звертання до **геттерів** з сховища (getters)

Загальна форма

this.\$store.getters.**геттер**

Фрагмент компонента

```
<template>
  <div>{{ $store.getters.doubleCount }}</div>
</template>
```

Сховище

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Передача даних у геттери

Загальна форма

this.\$store.getters.*геттер(параметри)*

Сховище

Фрагмент компонента

```
<template>
  ...<div>{{ $store.getters.getTodoById(2) }}</div>
</template>
```

```
getters: {
  // ...
  → getTodoById: (state) => (id) => {
    return state.todos.find(todo => todo.id === id)
  }
}
```

Пряме звертання до **геттерів** з сховища (getters)

Фрагмент компонента

```
<template>
  ...
  <div>
    <button @click="$store.dispatch('incrementCount')">
      Add
    </button>
    <button @click="$store.dispatch('decrementCount')">
      Subtract
    </button>
  </div>
</template>
```

Загальна форма

this.\$store.dispatch (*'action-метод', параметр*)

Сховище

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Приклад. Зробити менеджер задач з використанням Vuex

Додати задачу

Задача

Пріоритет

Список задач

Випити каву - 100	<input type="button" value="Done"/>	<input type="button" value="Delete"/>
Випити ще одну каву - 120	<input type="button" value="Done"/>	<input type="button" value="Delete"/>
Перевірити пошту - 5	<input type="button" value="Done"/>	<input type="button" value="Delete"/>
Зробити зарядку - 30	<input type="button" value="Done"/>	<input type="button" value="Delete"/>

Використання гелперів для *інтеграції* компонентів сховища у компонети

Для ітеграції (вставки) значень з state



mapState

Для ітеграції (вставки) гететрів



mapGetters

Для ітеграції (вставки) мутацій-методів



mapMutations

Для ітеграції (вставки) action-методів



mapActions

Зчитування значень з сховища у компонентах (state)

Для ітеграції (вставки) значень з state



mapState

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    }
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,

    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зчитування значень з сховища у компонентах (state)

----- Загальна схема -----

```
<template>
```

```
</template>
```

```
<script>
```

```
//1. Імпортуємо mapState  
import { mapState } from 'vuex'
```

----- Приклад -----

```
<template>
```

```
</template>
```

```
<script>
```

```
import { mapState } from 'vuex'
```

```
export default {
```

```
  ... name: 'App',
```

```
}
```

```
</script>
```

```
import { createStore } from 'vuex'  
  
// Create a new store instance.  
const store = createStore({  
  ... state() {  
    ... return {  
      ... count: 0,  
    },  
  },  
  getters: {  
    ... // getCount: (state) => state.count,  
    ... getCount: ({ count }) => count,  
  },  
  ... doubleCount(state) {  
    ... return state.count * 2  
  },  
  ... },  
  mutations: {  
    ... increment(state) {  
      ... state.count++  
    },  
    ... decrement(state) {  
      ... state.count--  
    },  
  },  
  ... actions: {  
    ... incrementCount(context) {  
      ... context.commit('increment')  
    },  
    ... decrementCount({ commit }) {  
      ... commit('decrement')  
    },  
  },  
})  
export default store
```


Зчитування значень з сховища у компонентах (state)

----- Загальна схема -----

```
<template>

</template>

<script>
//1. Імпортуємо mapState
import { mapState } from 'vuex'

export default {
  . . . . .

  //2. Додаємо властивості
  computed: {
    ... mapState([ 'властивість1',
                  властивість2], ...),
  },
  . . . . .
}
```

----- Приклад -----

```
<template>

</template>

<script>
import { mapState } from 'vuex'

export default {
  name: 'App',

  computed: {
    ...mapState(['count']),
  },
}
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,

    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зчитування значень з сховища у компонентах (state)

----- Загальна схема -----

```
<template>
//3. Використовуємо властивості як
звичайні обчислювальні властивості
. . . . .
<тег>{{ властивість1 }}</тег >
. . . . .
<тег :атрибут = "властивість2" />
. . . . .
</template>

<script>
//1. Імпортуємо mapState
import { mapState } from 'vuex'

export default {
. . . . .

//2. Додаємо властивості
  computed: {
    ... mapState([ 'властивість1',
                     властивість2 ], ...),
  },
. . . . .
}
</script>
```

----- Приклад -----

```
<template>
. . . <div>{{ count }}</div>
</template>

<script>
import { mapState } from 'vuex'

export default {
  name: 'App',

  computed: {
    ...mapState([ 'count' ]),
  },
}
</script>
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    doubleCount(state) {
      return state.count * 2
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зчитування значень з сховища у компонентах. Використ. геттерів у компонентах.

Для інтеграції (вставки) геттерів



mapGetters

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    }
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зчитування значень з сховища у компонентах. Використ. геттерів у компонентах.

----- Загальна схема -----

```
<template>
```

```
</template>
```

```
<script>
```

```
//1. Імпортуємо mapGetters
```

```
import { mapGetters } from 'vuex'
```

```
export default {
```

```
</script>
```

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

----- Приклад -----

```
<template>
```

```
</template>
```

```
<script>
```

```
import { mapGetters } from 'vuex'
```

```
export default {
```

```
  name: 'App',
```

```
</script>
```

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

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,

    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зчитування значень з сховища у компонентах. Використанням геттерів у компонентах.

----- Загальна схема -----

----- Приклад -----

```
<template>

</template>

<script>
//1. Імпортуємо mapGetters
import { mapGetters } from 'vuex'

export default {
  . . . . .

  //2. Підключаємо геттери
  computed: {
    ...mapGetters(['назва']),
  },
  . . . . .
}
</script>

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

```
<template>

</template>

<script>
import { mapGetters } from 'vuex'

export default {
  name: 'App',

  computed: {
    ...mapGetters(['getCount']),
  },
  . . . . .
}
</script>

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

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зчитування значень з сховища у компонентах. Використанням геттерів у компонентах.

----- Загальна схема -----

----- Приклад -----

```
<template>
//3. Використовуємо геттери як
звичайну обчислювальну властивість
. . . . .
<тег>{{ getter }}</тег>
. . . . .
<тег :атрибут = "getter" />
. . . . .
</template>

<script>
//1. Імпортуємо mapGetters
import { mapGetters } from 'vuex'

export default {
. . . . .

//2. Підключаємо геттери
  computed: {
    ...mapGetters(['назва']),
  },
. . . . .
}
</script>

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

```
<template>
  <div>{{ getCount }}</div>
</template>

<script>
import { mapGetters } from 'vuex'

export default {
  name: 'App',

  computed: {
    ...mapGetters(['getCount']),
  },
}
</script>

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

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    doubleCount(state) {
      return state.count * 2
    },
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})
export default store
```

Зміна значень у сховищі з компонентів. Використанням mutation-методів у компонентах.

Для ітеграції (вставки) мутацій-методів



mapMutations

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    }
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

----- Загальна схема -----

```
<template>
```

```
</template>
```

```
<script>
```

```
//1. Імпортуємо mapMutations  
import { mapMutations } from 'vuex'
```

```
</script>
```

----- Приклад -----

```
<template>
```

```
</template>
```

```
<script>
```

```
import { mapMutations } from 'vuex'
```

```
export default {
```

```
  name: 'App',
```

```
}
```

```
</script>
```

```
import { createStore } from 'vuex'  
  
// Create a new store instance.  
const store = createStore({  
  state() {  
    return {  
      count: 0,  
    },  
  },  
  getters: {  
    // getCount: (state) => state.count,  
    getCount: ({ count }) => count,  
  },  
  mutations: {  
    increment(state) {  
      state.count++  
    },  
    decrement(state) {  
      state.count--  
    },  
  },  
  actions: {  
    incrementCount(context) {  
      context.commit('increment')  
    },  
    decrementCount({ commit }) {  
      commit('decrement')  
    },  
  },  
})  
export default store
```


Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

----- Загальна схема -----

```
<template>

</template>

<script>
//1. Імпортуємо mapMutations
import { mapMutations } from 'vuex'

export default {
  . . . . .

  //2. Підключаємо методи-мутації
  methods: {
    ...mapMutations (['назва']),
  },
  . . . . .
}
```

----- Приклад -----

```
<template>

</template>

<script>
import { mapMutations } from 'vuex'

export default {
  name: 'App',

  methods: {
    ...mapMutations(['increment', 'decrement']),
  },
}
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

----- Загальна схема -----

```
<template>
//3. Використовуємо методи-мутації
як звичайні методи
    . . . . .
    <тег @подія="методи-мутації" />
    . . . . .
</template>

<script>
//1. Імпортуємо mapMutations
import { mapMutations } from 'vuex'

export default {
    . . . . .

//2. Підключаємо методи-мутації
  methods: {
    ...mapMutations (['назва']),
  },
  . . . . .
}
</script>
```

----- Приклад -----

```
<template>
  <div>{{ $store.state.count }}</div>
  <div>
    <button @click="increment">
      Add
    </button>
    <button @click="decrement">
      Subtract
    </button>
  </div>
</template>

<script>
import { mapMutations } from 'vuex'

export default {
  name: 'App',
  methods: {
    ...mapMutations(['increment', 'decrement']),
  },
}
</script>
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  doubleCount(state) {
    return state.count * 2
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

Для інтеграції (вставки) action-методів



mapActions

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    }
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,
  },
  mutations: {
    doubleCount(state) {
      return state.count * 2
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

```
<template>
```

```
</template>
```

```
<script>
```

```
//1. Імпортуємо mapActions  
import { mapActions } from 'vuex'
```

```
export default {
```

```
</script>
```

----- Приклад -----

```
<template>  
  <div>{{ getCount }}</div>  
</template>  
  
<script>  
import { mapGetters, mapActions } from 'vuex'  
  
export default {  
  name: 'App',  
  
  computed: {  
    ...mapGetters(['getCount']),  
  },  
  
  actions: {  
    incrementCount(context) {  
      context.commit('increment')  
    },  
    decrementCount({ commit }) {  
      commit('decrement')  
    },  
  },  
}
```

```
import { createStore } from 'vuex'  
  
// Create a new store instance.  
const store = createStore({  
  state() {  
    return {  
      count: 0,  
    },  
  },  
  getters: {  
    // getCount: (state) => state.count,  
    getCount: ({ count }) => count,  
  },  
  mutations: {  
    increment(state) {  
      state.count++  
    },  
    decrement(state) {  
      state.count--  
    },  
  },  
  actions: {  
    incrementCount(context) {  
      context.commit('increment')  
    },  
    decrementCount({ commit }) {  
      commit('decrement')  
    },  
  },  
})  
export default store
```

Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

```
<template>
```

```
</template>
```

```
<script>
```

```
//1. Імпортуємо mapActions
```

```
import { mapActions } from 'vuex'
```

```
export default {
```

```
  . . . . .
```

```
//2. Підключаємо action-методи
```

```
  methods: {  
    ... mapActions(['action-метод']),  
  },  
  . . . . .
```

```
}
```

```
</script>
```

----- *Приклад* -----

```
<template>
```

```
  <div>{{ getCount }}</div>
```

```
</template>
```

```
<script>
```

```
import { mapGetters, mapActions } from 'vuex'
```

```
export default {
```

```
  name: 'App',
```

```
  computed: {
```

```
    ...mapGetters(['getCount']),
```

```
  },
```

```
  methods: {
```

```
    ...mapActions(['incrementCount',  
                  'decrementCount']),
```

```
  },
```

```
}
```

```
</script>
```

```
import { createStore } from 'vuex'
```

```
// Create a new store instance.
```

```
const store = createStore({
```

```
  state() {
```

```
    return {
```

```
      count: 0,
```

```
    },
```

```
  getters: {
```

```
    // getCount: (state) => state.count,
```

```
    getCount: ({ count }) => count,
```

```
    doubleCount(state) {
```

```
      return state.count * 2
```

```
    },
```

```
  mutations: {
```

```
    increment(state) {
```

```
      state.count++
```

```
    },
```

```
    decrement(state) {
```

```
      state.count--
```

```
    },
```

```
  actions: {
```

```
    incrementCount(context) {
```

```
      context.commit('increment')
```

```
    },
```

```
    decrementCount({ commit }) {
```

```
      commit('decrement')
```

```
    },
```

```
  },
```

```
})
```

```
export default store
```

Зміна значень у сховищі з компонентів. Використанням action-методів у компонентах.

----- Приклад -----

```
<template>
//3. Використовуємо action-методи
як звичайні методи
    . . . . .
    <тег @подія="action-метод" />
    . . . . .
</template>

<script>
//1. Імпортуємо mapActions
import { mapActions } from 'vuex'

export default {
    . . . . .

//2. Підключаємо action-методи
methods: {
    ... mapActions(['action-метод']),
    . . . . .
}
</script>
```

```
<template>
  <div>{{ getCount }}</div>
  <div>
    <button @click="incrementCount">
      Add
    </button>
    <button @click="decrementCount">
      Subtract
    </button>
  </div>
</template>

<script>
import { mapGetters, mapActions } from 'vuex'

export default {
  name: 'App',

  computed: {
    ...mapGetters(['getCount']),
  },

  methods: {
    ...mapActions(['incrementCount',
      'decrementCount']),
  },
}
```

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state() {
    return {
      count: 0,
    },
  },
  getters: {
    // getCount: (state) => state.count,
    getCount: ({ count }) => count,

    doubleCount(state) {
      return state.count * 2
    },
  },
  mutations: {
    increment(state) {
      state.count++
    },
    decrement(state) {
      state.count--
    },
  },
  actions: {
    incrementCount(context) {
      context.commit('increment')
    },
    decrementCount({ commit }) {
      commit('decrement')
    },
  },
})

export default store
```

Приклад. Зробити менеджер задач з використанням Vuex та гелперів

Додати задачу

Задача

Пріоритет

Список задач

Випити каву - 100

Випити ще одну каву - 120

Перевірити пошту - 5

Зробити зарядку - 30

Список студентів

Система оцінювання

▼

Категорія

▼

ПІБ	Середній бал
Коваль Тарас	10
Григоренко Іван	9
Маркович Оля	11
Василеко Іра	12
Пилип Тарас	7
Баба Галя	35
Дід Степан	21

12

5

Відмінник

Хорошист

Трійочник

Двійочник

Блатник


Приклад. Розробити з використанням VueX

Віємо на нашому сайті

Список товарів

Хліб	24 грн.	<input type="button" value="Купити"/>
Молоко	31 грн.	<input type="button" value="Купити"/>
Ковбаса	350 грн.	<input type="button" value="Купити"/>
Шовдарь	950 грн.	<input type="button" value="Купити"/>
Пікниця	210 грн.	<input type="button" value="Купити"/>

Виберіть валюту 

Гривня 
Доллар

Корзина

Вода	15 грн.	Відмовитись
Ноутбук	529999 грн.	Відмовитись
Комплект моніторів	25000 грн.	Відмовитись
Інвертор	13000 грн.	Відмовитись
Акумулятор	11000 грн.	Відмовитись
Разом до оплати	102 014 грн.	Оплатити

Ще додаткові слайди за аналогічними схемами

Використання геттерів

4. Використовуємо підключені геттери як звичайні властивості

2. Імпортуємо mapGetters

3. Вказуємо список геттерів, які потрібно використати у цьому компоненті

1. Описуємо getters

```
test-app > src > App.vue > {} "App.vue" > script
You, seconds ago | 1 author (You)
1 <template>
2   <div>{{ getCount }}</div>
3
4 </template>
5
6 <script>
7
8   import { mapGetters } from 'vuex'
9
10  export default {
11    name: 'App',
12    components: {
13      HelloWorld,
14    },
15
16    computed: {
17      ...mapGetters(['getCount']),
18    },
19  }
20 </script>
21
22 <style>
23 #app {
24   font-family: Avenir, Helvetica, Arial, sans-serif;
25   -webkit-font-smoothing: antialiased;
```

```
test-app > src > store > JS index.js > [0] store
1 import { createStore } from 'vuex'
2
3 // Create a new store instance.
4 const store = createStore({
5   state() {
6     return {
7       count: 0,
8     }
9   },
10  getters: {
11    getCount: (state) => {
12      return state.count
13    },
14  },
15
16  mutations: {
17    increment(state) {
18      state.count++
19    },
20  },
21 })
22
23 export default store
24
```

EXPLORER

App.vue src M JS index.js store M

src > App.vue > {} "App.vue"

You, 4 minutes ago | 1 author (You)

```
1 <template>
2   <div>{{ getCount }}</div>
3   <div>
4     <button @click="incrementCount">Add</button>
5     <button @click="decrementCount">Subtract</button>
6   </div>
7 </template>
8
9 <script>
10 import { mapGetters, mapActions } from 'vuex'
11
12 export default {
13   name: 'App',
14
15   computed: {
16     ...mapGetters(['getCount']),
17   },
18
19   methods: {
20     ...mapActions(['incrementCount', 'decrementCount']),
21   },
22 }
23 </script>
24
25 <style lang="scss"></style>
26
```

5. Використовуємо action-методи

3. Імпортуємо mapActions

4. З використанням mapActions підключаємо action-методи

JS index.js store M

src > store > JS index.js > store > mutations > increment

You, 11 seconds ago | 1 author (You)

```
1 import { createStore } from 'vuex'
2
3 // Create a new store instance.
4 const store = createStore({
5   state() {
6     return {
7       count: 0,
8     }
9   },
10   getters: {
11     // getCount: (state) => state.count,
12     getCount: ({ count }) => count,
13
14     doubleCount(state) {
15       return state.count * 2
16     },
17   },
18   mutations: {
19     increment(state) {
20       state.count++
21     },
22     decrement(state) {
23       state.count--
24     },
25   },
26   actions: {
27     incrementCount(context) {
28       context.commit('increment')
29     },
30     decrementCount({ commit }) {
31       commit('decrement')
32     },
33   },
34 })
35 export default store
```

1. Описуємо мутацію

2. Описуємо action-метод

Зміна значень у сховищі state.
Використання action-методів

