# Vue.js Стартовий

## State manager: Vuex & Pinia

# Vue.js Стартовий

## Introduction


Кінаш Станіслав

Front-end dev

stanislav.kinash

stanislav.kinash



MCID: 9210561

## Тема уроку

# State manager: Vuex & Pinia

# План уроку

1. Для чого потрібен State Management.
2. Підключення Vuex. Огляд Vuex.
3. Підключення Pinia. Огляд Pinia.

# Для чого потрібен State Management

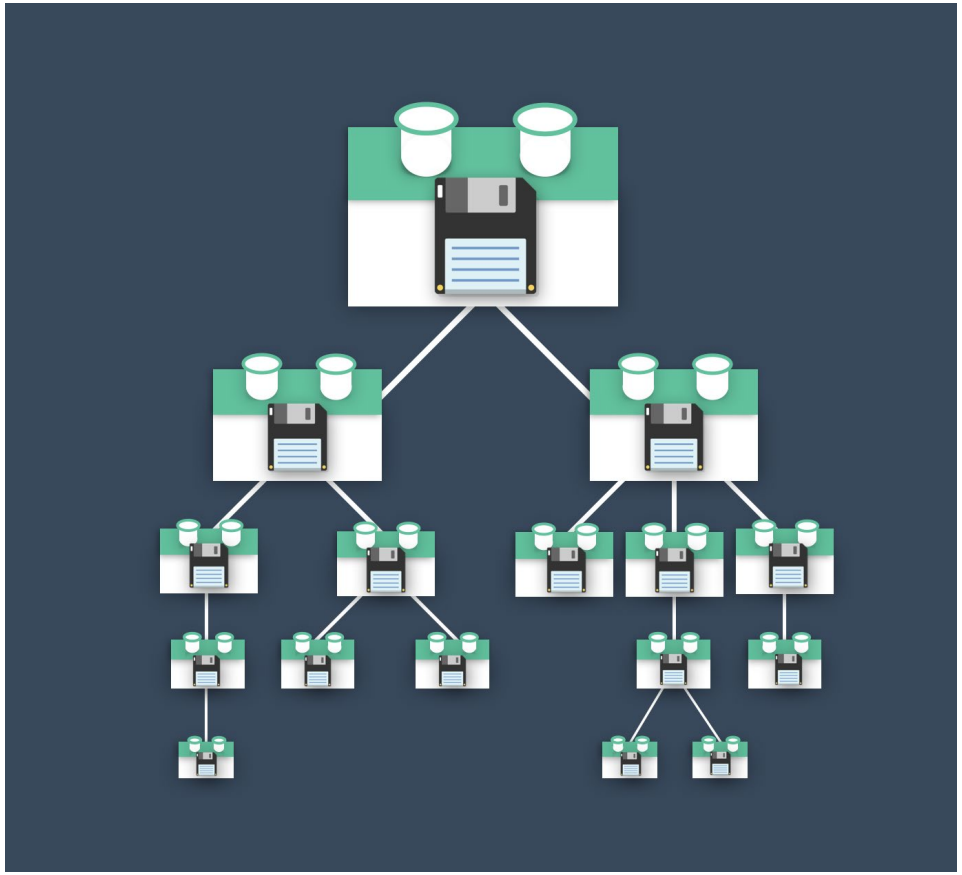
**Управління станом** є невід'ємною частиною створення складних вебдодатків, оскільки воно дозволяє зберігати та керувати станом вашої програми та обмінюватися ним між різними компонентами.

У Vue 3 ви можете використовувати вбудовані API **reactive** і **ref** для керування станом на рівні компонентів. Однак із збільшенням розміру і складності вашої програми централізоване та узгоджене керування станом може стати складним завданням, і саме тут у гру вступають бібліотеки керування станом.

# Для чого потрібен State Management

**Спрощує зв'язок між компонентами:** коли декільком компонентам потрібен доступ до одних і тих самих даних або стану, передача цих даних вгору та вниз по дереву компонентів може швидко стати складною та важкою для керування. Управління станом забезпечує централізований спосіб для компонентів обмінюватися даними, спрощуючи спілкування та полегшуючи міркування щодо вашої програми.

**Покращує продуктивність:** зберігаючи стан у централізованому розташуванні, бібліотеки керування станом, такі як Vuex, можуть оптимізувати оновлення стану та зменшити кількість повторних візуалізацій у вашій програмі. Це може призвести до покращення продуктивності та взаємодії з користувачем.



# Для чого потрібен State Management

**Забезпечує узгоджену обробку даних:** за допомогою керування станом ви можете забезпечити узгоджену обробку даних у своїй програмі. Визначивши, як дані мають оновлюватися та отримувати доступ в одному місці, ви можете уникнути непослідовної обробки даних і зменшити ймовірність помилок.

**Полегшує налагодження та тестування:** бібліотеки керування станом, такі як Vuex, часто постачаються з інструментами та плагінами, які полегшують налагодження та тестування вашої програми. За допомогою централізованого керування станом ви можете легше відстежувати помилки та перевіряти поведінку програми.

# Для чого потрібен State Management

**Добре масштабується з великими програмами:** у міру того, як ваша програма росте, ефективніше керувати станом стає важче. Бібліотеки керування станом, такі як Vuex, забезпечують масштабований спосіб керування станом, оскільки ваша програма стає більшою та складнішою, що полегшує її підтримку та розширення з часом.

# Підключення Vuex. Огляд Vuex.

Бібліотеки керування станом, як-от **Vuex**, забезпечують централізоване сховище для керування станом у додатку Vue, що полегшує керування й оновлення стану кількох компонентів. За допомогою **Vuex** ви можете визначити набір мутацій, які можна використовувати для оновлення стану передбачуваним і контрольованим способом, а також геттери, які можна використовувати для отримання обчислених властивостей на основі стану.

# Підключення Vuex. Огляд Vuex.

Ось приклад того, як ви можете використовувати Vuex для керування станом у програмі Vue 3:

Спочатку встановіть бібліотеку Vuex за допомогою обраного менеджера пакетів. Наприклад, якщо ви використовуєте npm, ви можете виконати таку команду:

```
npm install vuex
```

Далі у каталозі `src` створіть новий файл під назвою `store.js`. У цьому файлі імпортуйте функцію `createStore` з бібліотеки Vuex і визначте об'єкт початкового стану з деякими прикладами даних:

```
import { createStore } from 'vuex'

const store = createStore({
  state: {
    todos: [
      { id: 1, title: 'Do laundry', completed: false },
      { id: 2, title: 'Buy groceries', completed: false },
      { id: 3, title: 'Walk the dog', completed: true },
    ]
  }
})

export default store
```

# Підключення Vuex. Огляд Vuex.

У свій файл main.js імпортуйте об’єкт store та скористайтеся методом app.use, щоб додати його до програми Vue:

```
import { createApp } from 'vue'
import App from './App.vue'
import store from './store'

const app = createApp(App)
app.use(store)
app.mount('#app')
```

# Підключення Vuex. Огляд Vuex.

Тепер ви можете використовувати допоміжну функцію **mapState** від **Vuex**, щоб відобразити стан із сховища на обчислену властивість у вашому компоненті:

```
<template>
  <div>
    <h1>TODO App</h1>
    <ul>
      <li v-for="todo in todos" :key="todo.id">
        {{ todo.title }} - {{ todo.completed ? 'completed' : 'not completed' }}
      </li>
    </ul>
  </div>
</template>

<script>
import { mapState } from 'vuex'

export default {
  computed: {
    ...mapState(['todos'])
  }
}
</script>
```

# Підключення Vuex. Огляд Vuex.

Тепер цей компонент відображатиме список завдань із магазину **Vuex**.

Ви також можете визначити мутації в сховищі, які можна використовувати для оновлення стану. Наприклад, ви можете визначити **мутацію**, щоб перемикати властивість **completed** завдання:

```
const store = createStore({
    state: {
      todos: [
        { id: 1, title: 'Do laundry', completed: false },
        { id: 2, title: 'Buy groceries', completed: false },
        { id: 3, title: 'Walk the dog', completed: true },
      ]
    },
    mutations: {
      toggleTodoCompletion(state, todoId) {
        const todo = state.todos.find(todo => todo.id === todoId)
        todo.completed = !todo.completed
      }
    }
  })
```

# Підключення Vuex. Огляд Vuex.

Потім ви можете використати допоміжну функцію **`mapMutations`**, щоб зіставити мутацію **`toggleTodoCompletion`** із методом у вашому компоненті:

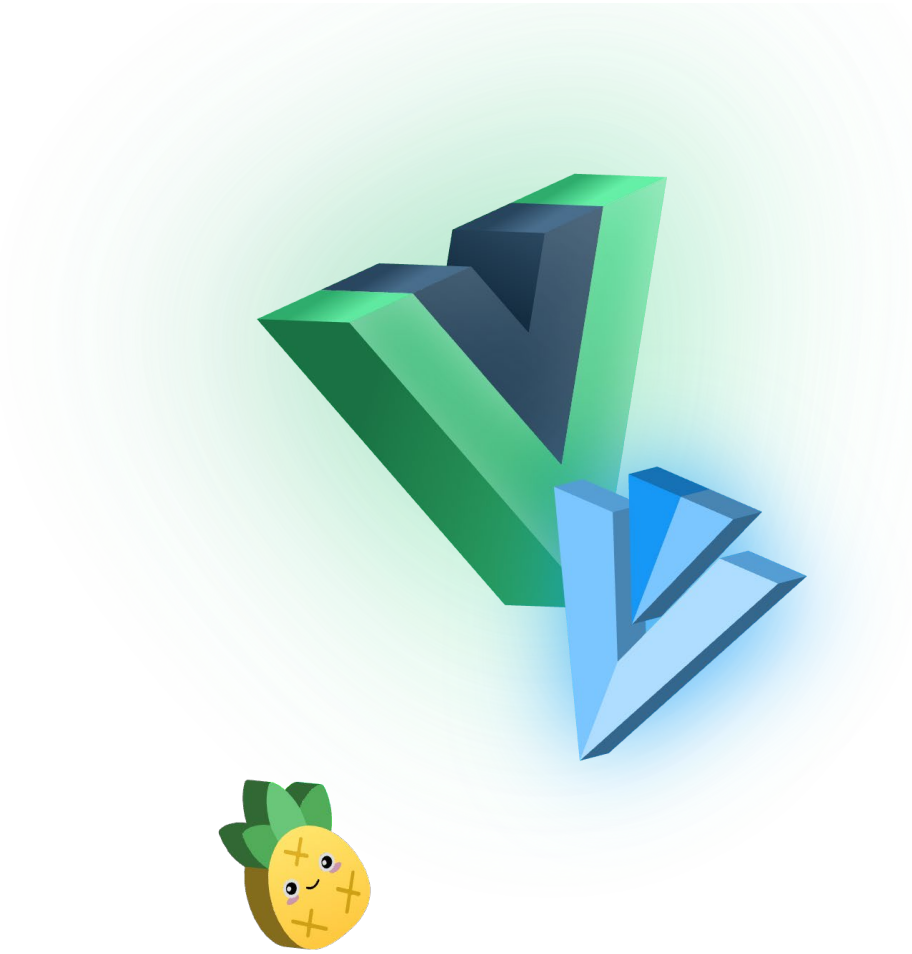
Цей компонент тепер відображає список завдань, кожне з яких має кнопку для перемикання статусу завершення. Після натискання кнопки викликається мутація `toggleTodoCompletion`, яка оновлює властивість `completed` відповідного завдання.

```
<template>
  <div>
    <h1>TODO App</h1>
    <ul>
      <li v-for="todo in todos" :key="todo.id">
        {{ todo.title }} - {{ todo.completed ? 'completed' : 'not completed' }}
        <button @click="toggleTodoCompletion(todo.id)">Toggle completion</button>
      </li>
    </ul>
  </div>
</template>

<script>
import { mapState, mapMutations } from 'vuex'

export default {
  computed: {
    ...mapState(['todos'])
  },
  methods: {
    ...mapMutations(['toggleTodoCompletion'])
  }
}
</script>
```

# Підключення Pinia. Огляд Pinia.

**Припустімо,** ми створюємо вебсайт електронної комерції за допомогою Vue 3 і нам потрібно керувати станом кошика для покупок користувача. Ми хочемо мати можливість додавати й видаляти товари з кошика, оновлювати кількість і обчислювати загальну вартість товарів у кошику.

Щоб керувати цим станом, ми використовуватимемо бібліотеку керування станом - **Pinia**.



# Підключення Pinia. Огляд Pinia.

Ось приклад використання **Pinia** для керування станом кошика для покупок.

Встановіть пакет `pinia`:

```
npm install pinia
```

Створіть **Pinia store:**

```
import { defineStore } from 'pinia'

export const useCartStore = defineStore({
  id: 'cart',
  state: () => ({
    items: []
  }),
  getters: {
    itemCount () {
      return this.items.reduce((count, item) => count + item.quantity, 0)
    },
    totalPrice () {
      return this.items.reduce((total, item) => total + item.quantity * item.price, 0)
    }
  },
```

# Підключення Pinia. Огляд Pinia.

Ось приклад використання **Pinia** для керування станом кошика для покупок.

Це створює магазин із масивом елементів для зберігання товарів у кошику. Він також визначає два **геттери**, **itemCount** і **totalPrice** для обчислення кількості товарів і загальної ціни товарів у кошику.

Нарешті, він визначає три дії: **addItem**, **removeItem** і **updateQuantity**, щоб додавати та видаляти товари з кошика, а також оновлювати кількість існуючих товарів.

```
  actions: {
    addItem (item) {
      const existingItem = this.items.find(existingItem => existingItem.id === item.id)
      if (existingItem) {
        existingItem.quantity += item.quantity
      } else {
        this.items.push(item)
      }
    },
    removeItem (itemId) {
      const index = this.items.findIndex(item => item.id === itemId)
      this.items.splice(index, 1)
    },
    updateQuantity (itemId, quantity) {
      const item = this.items.find(item => item.id === itemId)
      item.quantity = quantity
    }
  }
})
```

# Підключення Pinia. Огляд Pinia.

Використовуйте **store** у своїх компонентах:

Цей компонент використовує функцію **useCartStore** для створення реактивного об'єкта кошика, який можна використовувати для доступу до стану та дій, визначених у магазині.

```
<script>
import { useCartStore } from '@/store'

export default {
    setup () {
        const cart = useCartStore()

        return {
            $cart: cart
        }
    }
}
</script>
```

# Підключення Pinia. Огляд Pinia.

Використовуйте **`store`** у своїх компонентах:

Компонент відображає кількість товарів і загальну ціну товарів у кошику, а також надає кнопки для додавання та видалення товарів із кошика.

```
<template>
    <div>
        <h1>My Shopping Cart</h1>
        <p>Number of items: {{ $cart.itemCount }}</p>
        <p>Total price: {{ $cart.totalPrice }}</p>
        <ul>
            <li v-for="item in $cart.items" :key="item.id">
                {{ item.name }} - {{ item.price }} - {{ item.quantity }}
                <button @click="$cart.updateQuantity(item.id, item.quantity - 1)">-</button>
                <button @click="$cart.updateQuantity(item.id, item.quantity + 1)">+</button>
                <button @click="$cart.removeItem(item.id)">Remove</button>
            </li>
        </ul>
    </div>
</template>
```


TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA

# Перевірка знань

## TestProvider.com


TestProvider – це online-сервіс перевірки знань з інформаційних технологій. За його допомогою ви можете оцінити свій рівень та виявити слабкі місця. Він буде корисним як у процесі вивчення технології, так і для загальної оцінки знань IT-спеціаліста.

Перевірте, як ви засвоїли даний матеріал на TestProvider.com

Успішне проходження фінального тестування дозволить отримати відповідний Сертифікат.

# Vue.js Стартовий

## Дякую за увагу! До нових зустрічей!

Кінаш Станіслав
Front-end dev


MCID: 9210561