State Management

Nuxt provides the `useState` composable to create a reactive and SSR-friendly shared state across components.

`useState` is an SSR-friendly `ref` replacement. Its value will be preserved after server-side rendering (during client-side hydration) and shared across all components using a unique key.

👉 Read more in Docs > API > Composables > Use State.

👉 `useState` only works during `setup` or `Lifecycle Hooks`.

Because the data inside `useState` will be serialized to JSON, it is important that it does not contain anything that cannot be serialized, such as classes, functions or symbols.

Best Practices

🚨 Never define `const state = ref()` outside of `<script setup>` or `setup()` function.

Such state will be shared across all users visiting your website and can lead to memory leaks!

✅ Instead use `const useX = () => useState('x')`

Examples

Basic Usage

In this example, we use a component-local counter state. Any other component that uses `useState('counter')` shares the same reactive state.

app.vue

```
<script setup lang="ts">
const counter = useState('counter', () => Math.round(Math.random() * 1000))
</script>

<template>
  <div>
    Counter: {{ counter }}
    <button @click="counter++">
      +
    </button>
    <button @click="counter--">
      -
    </button>
  </div>
</template>
```

🕹️ Read and edit a live example in Docs > Examples > Features > State Management.

👉 Read more in Docs > API > Composables > Use State.

📘 To globally invalidate cached state, see `clearNuxtState`.

Advanced Usage

In this example, we use a composable that detects the user's default locale from the HTTP request headers and keeps it in a `locale` state.

composables/locale.ts

```
import type { Ref } from 'vue'

export const useLocale = () => useState<string>('locale', () => useDefaultLocale().value)

export const useDefaultLocale = (fallback = 'en-US') => {
```

```
  const locale = ref(fallback)
  if (process.server) {
    const reqLocale = useRequestHeaders()['accept-language']?.split(',')[0]
    if (reqLocale) {
      locale.value = reqLocale
    }
  } else if (process.client) {
    const navLang = navigator.language
    if (navLang) {
      locale.value = navLang
    }
  }
  return locale
}

export const useLocales = () => {
  const locale = useLocale()
  const locales = ref([
    'en-US',
    'en-GB',
    ...
    'ja-JP-u-ca-japanese'
  ])
  if (!locales.value.includes(locale.value)) {
    locales.value.unshift(locale.value)
  }
  return locales
}

export const useLocaleDate = (date: Ref<Date> | Date, locale = useLocale()) => {
  return computed(() => new Intl.DateTimeFormat(locale.value, { dateStyle: 'full' }).format(unref(
}
```

app.vue

```
<script setup lang="ts">
const locales = useLocales()
const locale = useLocale()
const date = useLocaleDate(new Date('2016-10-26'))
</script>

<template>
  <div>
    <h1>Nuxt birthday</h1>
    <p>{{ date }}</p>
    <label for="locale-chooser">Preview a different locale</label>
    <select id="locale-chooser" v-model="locale">
```

```
      <option v-for="locale of locales" :key="locale" :value="locale">
        {{ locale }}
      </option>
    </select>
  </div>
</template>
```

Read and edit a live example in Docs > Examples > Advanced > Locale.

Shared State

By using auto-imported composables we can define global type-safe states and import them across the app.

composables/states.ts

```
export const useCounter = () => useState<number>('counter', () => 0)
export const useColor = () => useState<string>('color', () => 'pink')
```

app.vue

```
<script setup lang="ts">
const color = useColor() // Same as useState('color')
</script>

<template>
  <p>Current color: {{ color }}</p>
</template>
```

Using third-party libraries

Nuxt **used to rely** on the Vuex library to provide global state management. If you are migrating from Nuxt 2, please head to the migration guide.

Nuxt is not opinionated about state management, so feel free to choose the right solution for your needs. There are multiple integrations with the most popular state management libraries, including:

- Pinia - the official Vue recommendation
- Harlem - immutable global state management

- XState - state machine approach with tools for visualizing and testing your state logic

Edit on Github