useCookie

Within your pages, components and plugins you can use `useCookie`, an SSR-friendly composable to read and write cookies.

```
const cookie = useCookie(name, options)
```

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

😌 `useCookie` ref will automatically serialize and deserialize cookie value to JSON.

Example

The example below creates a cookie called `counter`. If the cookie doesn't exist, it is initially set to a random value. Whenever we update the `counter` variable, the cookie will be updated accordingly.

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

<template>
  <div>
    <h1>Counter: {{ counter || '-' }}</h1>
    <button @click="counter = null">reset</button>
    <button @click="counter--">-</button>
    <button @click="counter++">+</button>
  </div>
```

```
</template>
```

Open on StackBlitz

Options

Cookie composable accepts several options which let you modify the behavior of cookies.

Most of the options will be directly passed to the **cookie** package.

`maxAge / expires`

`maxAge` Specifies the `number` (in seconds) to be the value for the `Max-Age` `Set-Cookie` **attribute**. The given number will be converted to an integer by rounding down. By default, no maximum age is set.

`expires` : Specifies the `Date` object to be the value for the `Expires` `Set-Cookie` **attribute**. By default, no expiration is set. Most clients will consider this a "non-persistent cookie" and will delete it on a condition like exiting a web browser application.

> 💡 **Note:** The cookie storage model specification states that if both `expires` and `maxAge` is set, then `maxAge` takes precedence, but not all clients may obey this, so if both are set, they should point to the same date and time!

If neither of `expires` and `maxAge` is set, the cookie will be session-only and removed when the user closes their browser.

`httpOnly`

Specifies the `boolean` value for the `HttpOnly` `Set-Cookie` **attribute**. When truthy, the `HttpOnly` attribute is set; otherwise it is not. By default, the `HttpOnly` attribute is not set.

> 💡 **Note:** Be careful when setting this to `true` , as compliant clients will not allow client-side JavaScript to see the cookie in `document.cookie` .

secure

Specifies the `boolean` value for the `Secure` `Set-Cookie` attribute. When truthy, the `Secure` attribute is set; otherwise it is not. By default, the `Secure` attribute is not set.

> 💡 **Note:** Be careful when setting this to `true`, as compliant clients will not send the cookie back to the server in the future if the browser does not have an HTTPS connection. This can lead to hydration errors.

domain

Specifies the value for the `Domain` `Set-Cookie` attribute. By default, no domain is set, and most clients will consider applying the cookie only to the current domain.

path

Specifies the value for the `Path` `Set-Cookie` attribute. By default, the path is considered the "default path".

sameSite

Specifies the `boolean` or `string` value for the `SameSite` `Set-Cookie` attribute.

- `true` will set the `SameSite` attribute to `Strict` for strict same-site enforcement.
- `false` will not set the `SameSite` attribute.
- `'lax'` will set the `SameSite` attribute to `Lax` for lax same-site enforcement.
- `'none'` will set the `SameSite` attribute to `None` for an explicit cross-site cookie.
- `'strict'` will set the `SameSite` attribute to `Strict` for strict same-site enforcement.

More information about the different enforcement levels can be found in the specification.

encode

Specifies a function that will be used to encode a cookie's value. Since the value of a cookie has a limited character set (and must be a simple string), this function can be used to encode a value into a string suited for a cookie's value.

The default encoder is the `JSON.stringify + encodeURIComponent`.

decode

Specifies a function that will be used to decode a cookie's value. Since the value of a cookie has a limited character set (and must be a simple string), this function can be used to decode a previously encoded cookie value into a JavaScript string or other object.

The default decoder is `decodeURIComponent` + destr.

> **Note:** If an error is thrown from this function, the original, non-decoded cookie value will be returned as the cookie's value.

default

Specifies a function that returns the cookie's default value. The function can also return a `Ref`.

watch

Specifies the `boolean` or `string` value for watch cookie ref data.

- `true` - Will watch cookie ref data changes and its nested properties. (default)
- `shallow` - Will watch cookie ref data changes for only top level properties
- `false` Will not watch cookie ref data changes.

Example 1:

```
<script setup lang="ts">
const user = useCookie(
  'userInfo',
  {
    default: () => ({ score: -1 }),
```

```
    watch: false
  }
)

if (user.value && user.value !== null) {
  user.value.score++; // userInfo cookie not update with this change
}
</script>

<template>
  <div>User score: {{ user?.score }}</div>
</template>
```

Example 2:

```vue
<script setup lang="ts">
const list = useCookie(
  'list',
  {
    default: () => [],
    watch: 'shallow'
  }
)

function add() {
  list.value?.push(Math.round(Math.random() * 1000))
  // list cookie not update with this change
}

function save() {
  if (list.value && list.value !== null) {
    list.value = [...list.value]
    // list cookie update with this change
  }
}
</script>

<template>
  <div>
    <h1>List</h1>
    <pre>{{ list }}</pre>
    <button @click="add">Add</button>
    <button @click="save">Save</button>
  </div>
</template>
```

Handling Cookies in API routes

Example:

```ts
export default defineEventHandler(event => {
  // Read counter cookie
  let counter = getCookie(event, 'counter') || 0
```

```
  // Increase counter cookie by 1
  setCookie(event, 'counter', ++counter)

  // Send JSON response
  return { counter }
})
```

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

Edit on Github