useNuxtData

`useNuxtData` gives you access to the current cached value of `useAsyncData`, `useLazyAsyncData`, `useFetch` and `useLazyFetch` with explicitly provided key.

Type

```ts
useNuxtData<DataT = any> (key: string): { data: Ref<DataT | null> }
```

Examples

Show stale data while fetching in the background

The example below shows how you can use cached data as a placeholder while the most recent data is being fetched from the server.

archive.vue

```ts
// We can access same data later using 'posts' key
const { data } = await useFetch('/api/posts', { key: 'posts' })
```

single.vue

```ts
// Access to the cached value of useFetch in archive.vue
const { data: posts } = useNuxtData('posts')

const { data } = await useFetch(`/api/posts/${postId}`, {
  key: `post-${postId}`,
```

```
  default: () => {
    // Find the individual post from the cache and set it as the default value.
    return posts.value.find(post => post.id === postId)
  }
})
```

Optimistic Updates

We can leverage the cache to update the UI after a mutation, while the data is being invalidated in the background.

todos.vue

```
// We can access same data later using 'todos' key
const { data } = await useFetch('/api/todos', { key: 'todos' })
```

add-todo.vue

```
const newTodo = ref('')
const previousTodos = ref([])

// Access to the cached value of useFetch in todos.vue
const { data: todos } = useNuxtData('todos')

const { data } = await useFetch('/api/addTodo', {
  key: 'addTodo',
  method: 'post',
  body: {
    todo: newTodo.value
  },
  onRequest () {
    previousTodos.value = todos.value // Store the previously cached value to restore if fetch fai

    todos.value.push(newTodo.value) // Optimistically update the todos.
  },
  onRequestError () {
    todos.value = previousTodos.value // Rollback the data if the request failed.
  },
  async onResponse () {
    await refreshNuxtData('todos') // Invalidate todos in the background if the request succeeded.
  }
})
```

Edit on Github