New Composition API

Nuxt Bridge implements composables compatible with Nuxt 3.

By migrating from `@nuxtjs/composition-api` to the Nuxt 3 compatible API, there will be less rewriting when migrating to Nuxt 3.

`ssrRef` and `shallowSsrRef`

These two functions have been replaced with a new composable that works very similarly under the hood: `useState`.

The key differences are that you must provide a *key* for this state (which Nuxt generated automatically for `ssrRef` and `shallowSsrRef`), and that it can only be called within a Nuxt 3 plugin (which is defined by `defineNuxtPlugin`) or a component instance. (In other words, you cannot use `useState` with a global/ambient context, because of the danger of shared state across requests.)

```
- import { ssrRef } from '@nuxtjs/composition-api'

- const ref1 = ssrRef('initialData')
- const ref2 = ssrRef(() => 'factory function')
+ const ref1 = useState('ref1-key', () => 'initialData')
+ const ref2 = useState('ref2-key', () => 'factory function')
  // accessing the state
  console.log(ref1.value)
```

Because the state is keyed, you can access the same state from multiple locations, as long as you are using the same key.

You can read more about how to use this composable in the Nuxt 3 docs.

ssrPromise

This function has been removed, and you will need to find an alternative implementation if you were using it. If you have a use case for `ssrPromise`, please let us know via a discussion.

onGlobalSetup

This function has been removed, but its use cases can be met by using `useNuxtApp` or `useState` within `defineNuxtPlugin`. You can also run any custom code within the `setup()` function of a layout.

```
- import { onGlobalSetup } from '@nuxtjs/composition-api'

- export default () => {
-   onGlobalSetup(() => {
+ export default defineNuxtPlugin((nuxtApp) => {
+   nuxtApp.hook('vue:setup', () => {
      // ...
    })
- }
+ })
```

useStore

In order to access Vuex store instance, you can use `useNuxtApp().$store`.

```
- import { useStore } from '@nuxtjs/composition-api`
+ const { $store } = useNuxtApp()
```

useContext and withContext

You can access injected helpers using `useNuxtApp`.

```
- import { useContext } from '@nuxtjs/composition-api`
+ const { $axios } = useNuxtApp()
```

> 👉 `useNuxtApp()` also provides a key called `nuxt2Context` which contains all the same properties you would normally access from Nuxt 2 context, but it's advised *not* to use this directly, as it won't exist in Nuxt 3. Instead, see if there is another way to access what you need. (If not, please raise a feature request or discussion.)

`wrapProperty`

This helper function is not provided any more but you can replace it with the following code:

```
const wrapProperty = (property, makeComputed = true) => () => {
  const vm = getCurrentInstance().proxy
  return makeComputed ? computed(() => vm[property]) : vm[property]
}
```

`useAsync` and `useFetch`

These two composables can be replaced with `useLazyAsyncData` and `useLazyFetch`, which are documented in the Nuxt 3 docs. Just like the previous `@nuxtjs/composition-api` composables, these composables do not block route navigation on the client-side (hence the 'lazy' part of the name).

> Note that the API is entirely different, despite similar sounding names. Importantly, you should not attempt to change the value of other variables outside the composable (as you may have been doing with the previous `useFetch`).

> The `useLazyFetch` must have been configured for Nitro.

Migrating to the new composables from `useAsync`:

```
<script setup>
- import { useAsync } from '@nuxtjs/composition-api'
- const posts = useAsync(() => $fetch('/api/posts'))
```

```
+ const { data: posts } = useLazyAsyncData('posts', () => $fetch('/api/posts'))
+ // or, more simply!
+ const { data: posts } = useLazyFetch('/api/posts')
</script>
```

Migrating to the new composables from `useFetch`:

```
<script setup>
- import { useFetch } from '@nuxtjs/composition-api'
- const posts = ref([])
- const { fetch } = useFetch(() => { posts.value = await $fetch('/api/posts') })
+ const { data: posts, refresh } = useLazyAsyncData('posts', () => $fetch('/api/posts'))
+ // or, more simply!
+ const { data: posts, refresh } = useLazyFetch('/api/posts')
  function updatePosts() {
-   return fetch()
+   return refresh()
  }
</script>
```

useMeta

In order to interact with `vue-meta`, you may use `useNuxt2Meta`, which will work in Nuxt Bridge (but not Nuxt 3) and will allow you to manipulate your meta tags in a `vue-meta`-compatible way.

```
<script setup>
- import { useMeta } from '@nuxtjs/composition-api'
  useNuxt2Meta({
    title: 'My Nuxt App',
  })
</script>
```

You can also pass in computed values or refs, and the meta values will be updated reactively:

```
<script setup>
const title = ref('my title')
useNuxt2Meta({
  title,
})
title.value = 'new title'
```

```
</script>
```

> Be careful not to use both `useNuxt2Meta()` and the Options API `head()` within the same component, as behavior may be unpredictable.

Nuxt Bridge also provides a Nuxt 3-compatible meta implementation that can be accessed with the `useHead` composable.

```
<script setup>
- import { useMeta } from '@nuxtjs/composition-api'
  useHead({
    title: 'My Nuxt App',
  })
</script>
```

You will also need to enable it explicitly in your `nuxt.config`:

```
import { defineNuxtConfig } from '@nuxt/bridge'
export default defineNuxtConfig({
  bridge: {
    meta: true
  }
})
```

This `useHead` composable uses `@unhead/vue` under the hood (rather than `vue-meta`) to manipulate your `<head>`. Accordingly, it is recommended not to use both the native Nuxt 2 `head()` properties as well as `useHead`, as they may conflict.

For more information on how to use this composable, see the Nuxt 3 docs.

Explicit Imports

Nuxt exposes every auto-import with the `#imports` alias that can be used to make the import explicit if needed:

```
<script setup lang="ts">
import { ref, computed } from '#imports'

const count = ref(1)
```

```
const double = computed(() => count.value * 2)
</script>
```

Disabling Auto-imports

If you want to disable auto-importing composables and utilities, you can set `imports.autoImport` to `false` in the `nuxt.config` file.

nuxt.config.ts

```
export default defineNuxtConfig({
  imports: {
    autoImport: false
  }
})
```

This will disable auto-imports completely but it's still possible to use explicit imports from `#imports`.

Edit on Github