**Lekcja 31: Praca z kluczami**

Zdefiniowaliśmy obiekt, który pozwala na dowolny klucz typu string, o ile typem każdej właściwości jest URL. Dzięki temu wiemy już, że możemy przekazywać tylko obiekty, które mają prawidłowy kształt, bez narzekania kompilatora. Jednak po wybraniu odpowiedniego formatu nadal możemy przekazać do funkcji każdy ciąg, nawet jeśli format może nie istnieć.

```
declare const videos: VideoFormatURLs

loadFile(videos, 'format4k')

// 4K not available

// TypeScript doesn't squiggle
```

Oczywiście możemy zrobić to lepiej.

**Powiązane parametry typu**

Chcemy tylko przekazać klucze jako drugi argument, które są faktycznie dostępne w obiekcie. W nieogólnym świecie zrobilibyśmy coś takiego:

```
function loadVideoFormat(

fileFormats: VideoFormatURLs,

format: keyof VideoFormatURLs

) {

// You know

}
```

To samo dotyczy ogólnych parametrów typu:

```
type URLObject = {

[k: string]: URL

}

function loadFile<Formats extends URLObject>(

fileFormats: Formats,

format: keyof Formats

) {

// The real work ahead

}
```

To już daje nam świetne narzędzia. Teraz możemy wprowadzić tylko klucze, które są częścią obiektu, który przekazujemy jako pierwszy parametr:

```
// 'format4k' is not available

loadFile(video, 'format4k')
```

keyof Formats, po zastąpieniu przez VideoFormatURLs daje „format360p” | "format480p" | "format720p" | „format1080p”. Format, który przekazujemy dla drugiego argumentu, musi należeć do tego typu unii. Rzućmy okiem na treść funkcji i zróbmy bardzo prostą implementację. Uzyskujemy dostęp do adresu URL, pobieramy z niego dane i zwracamy obiekt, który mówi nam, jaki format załadowaliśmy i czy ładowanie się powiodło. Rzeczywista implementacja zawierałaby znacznie więcej szczegółów, ale to wszystko, czego potrzebujemy, aby zobaczyć, co się dzieje na poziomie typu. Ponieważ używamy funkcji async fetch, przekształcamy loadFile, aby był również funkcją async.

async function loadFile<Formats extends URLObject>(

fileFormats: Formats,

format: keyof Formats

) {

// Fetch the data

const data = await fetch(fileFormats[format].href)

return {

// Return the format

format,

// and see if we get an OK response

loaded: data.response === 200

}

}

Zobaczmy, co otrzymamy w zamian. Dzięki wnioskowaniu o typie zwracanym typem loadFile jest Promise <{format: keyof Frmats. loaded : boolean}>. Promise to typ ogólny, co nie powinno już dziwić. A właściwość format w naszej wartości zwracanej jest parametrem typu ogólnego, który zdefiniowaliśmy w naszej funkcji. Użyjmy naszej funkcji z substytutami.

const result = await loadFile (filmy, „format1080p”)

await rozpakowuje Promise <>, więc możemy zobaczyć, że rzeczywiste wartości zwracane są z loadFile. result jest typu {format: "format360p" | "format480p" | "format720p" | "format1080p", loading: boolean}. Jak się spodziewamy, jako zwrot otrzymujemy keyof VideoFormatURL. Ale czy nie powinniśmy wiedzieć więcej? Jawnie przekazujemy „format1080p” jako drugi argument. Zawęziliśmy już związek poprzez użycie pojedynczego typu wartości. Dlaczego wynik nie może być typu {format: "format1080p", loading: boolean}?

Możemy to osiągnąć, dodając drugi parametr typu do naszej deklaracji ogólnej, taki, który pokazuje związek z pierwszym, ale działa jako własny typ po zadeklarowaniu, na przykład:

function loadFile<

Formats extends URLObject,

Key extends keyof Formats

>(fileFormats: Formats, format: Key) {

const data = await fetch(fileFormats[format].href)

return {

format,

loaded: data.response === 200

}

}

Drugi parametr typu Key jest podtypem keyof Formats, pierwszym parametrem typu. Ciekawa część dzieje się teraz, kiedy zaczynamy pisać napisy:

loadFile (video, 'format1080p') // OK!

video jest typu VideoFormatURLs. VideoFormatURL jest podtypem obiektu URLObject, więc sprawdzanie typu przebiega pomyślnie a Formats można zastąpić. Teraz Key musi być podtypem keyof Formats. „format1080p" jest podtypem keyof Formats, więc sprawdzanie typu przebiega pomyślnie i Key można zastąpić. Teraz zablokowaliśmy i zastąpiliśmy dwa typy:

7. Formaty to VideoFormatURLs

8. Key to „format1080p"

Zgadza się. Teraz, gdy przekazaliśmy ciąg literału, parametr typu przyjmuje literał, typ wartości, co oznacza, że po wykonaniu tej funkcji i spojrzeniu na wynik możemy być pewni, że format result.format to „format1080p":

const result = await loadFile(videos, "format1080p")

if(result.format !== "format1080p") {

// result.format is now never!

throw new Error("Your implementation is wrong")

}

Aby upewnić się, że wdrażamy również właściwą rzecz, definiujemy typ zwracany dla funkcji loadFile, w której spodziewamy się pojawienia się typu Key.

type URLObject = {

[k: string]: URL

}

type Loaded<Key> = {

format: Key,

loaded: boolean

}

async function loadFile<

```
Formats extends URLObject,
Key extends keyof Formats
>(fileFormats: Formats, format: Key):
Promise<Loaded<Key>> {
const data = await fetch(fileFormats[format].href)
return {
format,
loaded: data.response === 200
}
}
```

Wszystko opakowane w ogólną Promise, ponieważ jesteśmy asynchroniczni.