**Stop using {} in Typescript**

In the world of TypeScript, it’s pretty common to use ‘**Object**’ or ‘**{}**’ as a type when we’re expecting an object but aren’t quite sure about the object’s ins and outs. It’s like getting a mystery package — you know it’s there, but you’re not entirely sure what’s inside. But here’s the twist: while this practice might seem handy at first glance, relying too much on these generic object types can lead to some unexpected issues down the road. Let’s dive in and explore why it might be time to hit the brakes on using these ambiguous types in your TypeScript code.

**Understanding the problem of using Object or {}**

Whenever we have a variable in Typescript storing any object but aren’t sure about the object structure, we often use type Object for {} as shown below

type Param = Object;

OR

type Param = {};

and using it at various places like function parameters:

function myFunc(obj:Param){  
 console.log(params);  
}

But this becomes a problem, as we know in Javascript an Object is the base of everything, hence allowing values like string, date, boolean, etc to be passed without throwing a Typescript error, as shown below consoles.

myFunc({name: 'John', age: 30});  
myFunc('abc');  
myFunc(123);  
myFunc(true);  
myFunc([1,2,3]);  
myFunc(new Date());  
myFunc(() => {});  
myFunc({});



Now that we’ve glimpsed the hiccups caused by using Object or {}, it’s time to roll up our sleeves and explore a few different paths to solve these snags. By finding alternatives, we can pave the way for smoother, more predictable code.

**Solution-1: using Record**

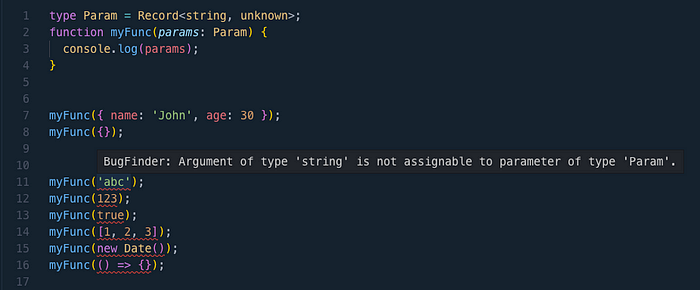
We can use **Record** in Typescript to solve this problem. Record takes 2 types one for the key and another for the value as:

type Param = Record<string, unkown>

Here we can see <string, unknown> is passed to Record which means the keys will be of type string and values are marked unknown.

Let us see using the record in the above example myFunc

type Param = { [key: string]: unknown };  
function myFunc(params: Param) {  
 console.log(params);  
}  
  
  
myFunc({ name: 'John', age: 30 });  
myFunc('abc');  
myFunc(123);  
myFunc(true);  
myFunc([1, 2, 3]);  
myFunc(new Date());  
myFunc(() => {});  
myFunc({});



Here we can see Typescript starts complaining out type for values like string, number, boolean, etc.

**Solution-2: using index**

Another approach is using the **index**where we can define the individual types for the key and value.  
Let’s say we want to use type **string** for **keys** and **unknown** for **values** then we can define our param as:

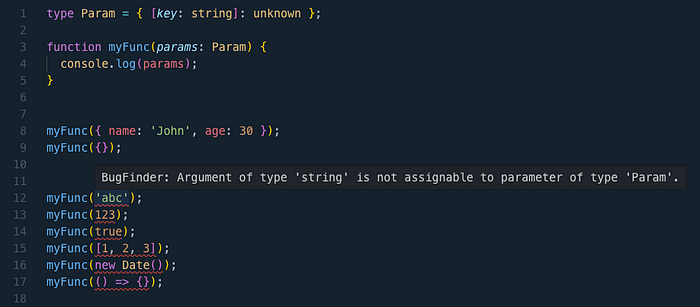
type Param = {  
 [index:string]: unknown  
}

*Note: index here is a placeholder and we can use any other terms like key, property, id, etc like:  
type Param = { [key: string]: unknown };*

now let us see what happens if use it in our example code with myFunc

type Param = { [index: string]: unknown };  
  
function myFunc(params: Param) {  
 console.log(params);  
}  
  
  
myFunc({ name: 'John', age: 30 });  
myFunc('abc');  
myFunc(123);  
myFunc(true);  
myFunc([1, 2, 3]);  
myFunc(new Date());  
myFunc(() => {});  
myFunc({});

In the above example, we defined the type Param using the index and this is how Typescript starts complaining when you pass params like string, number, boolean, etc.



For more such articles checkout my blogs at [varunsukheja.com](http://varunsukheja.com/)