* -: The next step now of course is ***to output our posts .***
* Before we do that, however, let me bring your attention to a tiny problem we have here or to a spot where we can improve our code .
* If we hover over posts here in your ID, you probably see the type of posts and you see that this is of type any .

Text

Description automatically generated

* That of course means the TypeScript is not able to detect how our posts look like, the format of our post .
* Cause here posts array all sorts of type any, because the only thing TypeScript sees is that we add the ID key here .
* But other than that the object we push is totally unknown to it because it doesn't know the format of our response data .
* One way of doing it would be to assign a type here on the argument we're getting in Map .
* So we could say, yeah, we know that this actually will be an object where we have a randomly generated string and now this syntax can look really weird in TypeScript but in the end you use a placeholder property name for that by using square brackets here .
* And then you add key colon string in there which simply means this is any string key we have here on that object .
* We don't know the exact property name because it's randomly generated or not randomly but it's generated by Firebase .
* And we do have many different properties but we know it will be a property that can be interpreted as a string .

Text

Description automatically generated

* And now the value held by that property is our actual post data, right? It's the same format we have here .
* Now, therefore, to simplify it as a bid for us, we can of course add a new model .
* Let's say a post model here where I export that interface and I'll name this post and this will just have that format here, title and content and actually will optionally all to have an ideal .
* Let's add a quest mark here to make clear that this is a string but it's optional and now we can use that here .
* So there we use post, we just need to import that .
* So let's import post from our post model file here .
* So from dot slash post dot model .

Text

Description automatically generated

* And now we know here we have that randomly generated ID but that will then hold a post as a value .
* That's that nested element I showed you in the developer tools earlier .

Text

Description automatically generated

* With that now TypeScript suddenly knows what's inside of response data .
* And now we can also set the type here on our post array and say, yeah, that will be a post array which you set by using this type assignment the name of the type, and then square brackets to indicate that it will be an array of these types .
* This now is improved code because now we can clearly understand what's inside of posts array and we get better auto-completion .
* For example, if I try to access the first element, you see I can now access ID content and title because I now know that it has this format and that therefore is a recommended practice .
* However, with the angler HDP Client, there is a more elegant way of assigning the type .
* You don't need to do this here on the argument in Map or in Subscribe or whatever you are using to access your data .
* Instead, the get method is a so-called generic method .
* This means you can add angled brackets here and between the angled brackets you store the type which this response will actually return as a body once it's done .
* So it's the response body type, which you store here and that will then automatically be handled by the angler HDP Client and TypeScript understands this and now knows that the response data will have this type format as you can tell .
* And this is also available on post requests, it's available on all requests .

Text

Description automatically generated

* You can use these angled brackets to add this extra piece of information, which is totally optional, but recommended and helpful about the data you are getting back .
* And now here you know it'll be a JavaScript object where you have a name property and the value of that will be a string .
* Because if you remember when you send a post this is a new post, really new, the data you get back is exactly such an object, name, property and then a random string .
* And now we're making this really clear here so that we also notice in our subscribe function and if we had code that depends on the types we get better auto completion and avoid unnecessary TypeScript errors .
* So this is optional but recommended .
* You can define the response data types here on these generic HDP wear methods .
* Learning reminders Calendar events