* Instructor: Thus far our requests always succeeded.
* But of course when interacting with servers, things can go wrong.
* You might not have an internet connection or you're sending a request with incorrect data somehow because you have a bug in your program or whatever it is or your server fails.
* Maybe it's offline or there's an error on the server or you are not authenticated.
* There are dozens of things that can go wrong and we can simulate such a thing that can go wrong by going to Firebase, and there in the database, let's go to Rules.
* And remember that when we created that database, we enabled test mode which essentially means that read and write access to this database is granted to everyone.
* You don't need to authenticate yourself.
* Now, we will add authentication in the authentication module later in the course and therefore for now we open this up to everyone.
* Now, if I lock this down though so if I say you're not allowed, to read for example you can still write but you're not allowed to read.
* And by setting this to false, there is no condition under which you would ever be allowed, not even if you are authenticated.
* So if I lock this down, what happens is that if I now try to fetch posts here, you see we get an error and that of course happens, errors occur.
* But it is important how we react to that error.
* For now, for example, here we stay in this loading state and that is not a great user experience because now we as a user have no idea that there was an error and that something failed.
* Therefore, proper error handling matters.
* Now, how can we handle errors? There are different ways of doing that.
* Let me demonstrate the first possible way here, "onFetchPosts" fails because, well, we have an error.
* Thus far, we only passed one function to subscribe and that is the function that fires when new data is emitted.
* Now, as you learned in the observables section you can pass more arguments to subscribe and the second argument is a function that triggers whenever an error is thrown and there we will get the error object as an argument.
* And now here we can do something to handle that error.
* To do something, to provide a better user interface or a better user experience.
* And that often starts with a better user interface.
* So, for example here what we could do is, we could display an error message instead of loading.
* So down there where we display loading or the content we loaded, we could actually introduce a new property error and set this to null initially.
* So, we have no error initially, if this is not now.
* So, if we do have an error and we'll add logic to change that soon, I want to add a div here with the class as alert and alert danger which are bootstrap classes, where I simply have let's say, h1 tag, an error occurred.
* And below that I wanna output error because error in my case, in this application here could just be an error message.
* And of course it's up to you how you exactly manage that and what you store in there.
* Now, the idea is that this here, this div is only shown if we have an error.
* So if error is true-ish, which it is not if it is null.
* Initially it's null and therefore, it's treated as a false value and therefore initially this will not show up.
* But as soon as we set this to some string it becomes true-ish and then this will show up.
* This all means that I only want to show loading if we are fetching and we're not having an error.
* Because if we have an error, I never wanna show loading.
* I want to show my alert here.
* Now, the remaining step is to set that error and of course I want to do this here in my error handling function.
* And I will simply set this error, "equal.message" because by default error objects have a message.
* The question is if that message is helpful but at least we can set this message for now.
* Now, important, of course we are fetching post not just here but all "ngOnInit", so there we should add the second argument as well.
* And with that, let's have a look.
* Let's wait for that to reload.
* And here it is already.
* Now, of course you can see, this is the error message we're getting.
* Might not be too useful.
* You can definitely tweak that, you can show your own error message.
* You can dive into that in more details and if you for example, need more information from the error response.
* Well, let's have a look at the error we got here.
* Let me save that.
* Now if a fetch posts, this is getting locked and you see, what I log here is indeed a full error response object.
* Now there you see that message we accessed, but you also see, you got, for example, another error key with yet another error key that has maybe a more helpful error message.
* This however is not guaranteed to exist.
* This only exists because our error here is an error response.
* So, the fire base server sending back a customized error response.
* Now the important thing to memorize about that however is, you will get that HTTP error response object by angular and it will have an error key.
* But the detailed content of what's in there, depends on the API you're talking to.
* Fire base gives you an object with another error key and that permission denied message.
* Your own API might not be sending this or might be sending different data.
* So, it's always important to understand which API you are working with and what this API sends back in the case of a success message or in the case of an error.
* If you need more information about the error though and not just that generic message then you can dive into that error object you're getting, which is that object we're logging here and there you see, you can also get information about the headers.
* You can get the exact status code that was thrown