* Instructor: The code we have thus far is not bad and it does its job.
* And if our application were as simple as it is here, it would be perfectly fine.
* But in bigger applications and maybe even in this one, we already have a quite big component with a lot of code that is only indirectly related to the user interface.
* Of course, we wanna send a request when a button is clicked and we wanna display that response data, but for example, transforming the result, transforming the data, we can do it here, it's not inherently bad, but it is a nice practice to outsource that into services, so that services are the parts in your Angular application that do the heavy lifting, the dirty work and your components are relatively lean, as lean as they can be, and are mostly concerned with template related work.
* So with things like there is fetching property and so on.
* And therefore, let's create a new service now, the posts service here.
* And I'll create it by exporting a class PostsService, and we need to provide that, and for that we can use two approaches.
* We can use @Injectable, which you need to import from @angular/core.
* And there we can pass an object and set provided into root, which is the more modern and recommended approach.
* But you could also provide it here in the providers array of the app module, and this also wouldn't be bad or a problem.
* Here, however I will use this approach.
* Now what's my idea behind that service? In this service, I want to have my HTTP request methods.
* So the methods that do the HTTP requests.
* And I only want to get the responses or the message whether we're done with the request in my front end.
* So therefore, in this service, I will add a new method, createAndStorePost, you can of course name this however you want.
* And there I expect to get a title which is a string, and the content which is a string as an argument, let's say.
* And here I wanna send my HTTP request.
* Now of course, I also want to be able to fetch my post, so we'll add fetchPosts here.
* And there, I also want to send a request.
* Now of course, I only want to move my code over.
* That's in the end what I want to do here.
* So let's grab that HTTP request here from onCreatePost and let's move it into the PostsService here.
* Now we're sending the HTTP request here.
* However, for this to work, we need to inject the HTTP service into this PostsService.
* So let's add a constructor, and with the help of that constructor, we inject the HTTP client, and we store it in a property named http.
* And the HTTP client is imported from @angular/common/http.
* Now of course, here we have no post data property anymore.
* I can create that here though, and this will in the end be of type Post and therefore it will have a title and so on but we first of all need to import our own post model.
* So let's import Post from ./post.model.
* And now to be a real post, we need to set a title property here which is equal to our title we're getting as a argument, and we need to set a content property which is equal to the content we're getting as a argument.
* So the right set of the colon refers to these arguments here.
* Now that's the post data we're sending.
* And now we can call createAndStorePost to send that request.
* So back in the app component, I now need to inject my own PostsService, private postsService like this, and make sure to add that import path here at the top.
* You need to import PostsService from there.
* And now in onCreatePost, I simply call this.postService.createAndStorePost.
* I get an error here because I need to pass some data to that method.
* I need to pass the title and the content.
* Now I get my post data here so I access postData.title, and my second argument I send is postData.content.
* And with that, we should be able to create new posts.
* Let's add a new post here, the best Angular course, which this one hopefully is.
* I'm of course interested in your feedback.
* So I hope so, sent that post.
* Now we have no logic to automatically update our list of posts down there but that's fine.
* We can click fetch posts again and now we see our post here.
* So that now still works but now the logic was outsourced into the PostsService.
* Let's do the same for fetching posts.
* Let's grab this logic here and we'll have a problem with isFetching but I'll come back to this.
* Let's grab this logic here and move it into fetchPosts in the PostsService like that.
* Now let's remove isFetching and let's remove this.loadedPosts because we have, we don't have these properties here and we don't really need them here in the PostsService.
* isFetching and the loadedPosts clearly belong into our app component and not into the service because the component is where we use them, where we want to display them or where we need them to change what we display in the template.
* Let's focus on the rest of this code for now.
* We're missing the map operator so let's make sure we import that here.
* Let's import map from rxjs/operators.
* And with that we have a request that should work but we'll still need to fine tune this later.
* At the moment, we can get rid of that private method here, of course, and instead where we previously called the this.fetchPosts, we can now use the postsService and call fetchPosts there.
* So we do that here in onFetchPosts and in ngOnInit.
* However, what you will see, of course is that now we have no posts available, because whilst we're running that code in the service and indeed you can see that if I click fetch posts a HTTP request is sent and we are fetching the posts.
* Whilst all of that is happening, we're not seeing the post here because we lost the connection between the data we fetched in the service and our template.
* Let's fix that.