Eugen: Welcome to this lesson out of Learn Spring, where we're going to continue the work we started previously by taking our naive controller implementation to the next level. All right, let's jump right into the code and let's get started. Let's, of course, open up our project controller. And now that our web support is starting to take shape, let's do some simple CRUD in our application. First of all, our current find one [00:00:30] implementation here is, as I was just seeing, naive. It simply returns some random data when what it actually should be doing is go in the service layer and use that to retrieve our project. So, that's our first step here.

[00:01:00] All right, so we injected our project service here, and of course we're using the find by ID implementation out of the service. However, notice that we need the ID of the project that we're looking for, since that is of course what the method is retrieving by. So, how do we get that? Well, we can have that as a parameter right here in the method signature.

And because that return's an optional, for [00:01:30] now, we're going to simply do a get on it. We're going to, of course, improve that later. But for now, that's okay. However, the real question here is where is that long parameter coming from? That should be coming from the mapping. However, our current mapping is just mapped to slash one. That was, of course, okay for our naive implementation. But right now, we need to change that.

Finally, [00:02:00] in order for this to get mapped as a path component, we need to use an annotation on the parameter. This is yet another annotation out of Spring MVC. This is helping with the mapping. So, that is what bridges the method signature here with the actual resolution of the ID. We can now include any long ID in the URL and this is going to resolve it. All right, so now that we [00:02:30] have our operation mapped, let's run the application. Let's open up a client. We're going to be using Postman for this and let's consume it from the client side.

All right, so with Postman open, let's open up our request here. We have it prepared, so that we don't have to set it up all over again every time we use it. Let's now hit our application and see what type of response we get back. All right, so we're getting back a project [00:03:00] JSON. That's all we care about right now. We're naturally going to explore this side of things, the request side of things later on. But right now, what we care about is that we're able to consume the data through that operation. But we weren't actually able to hit slash projects slash one last time as well.

Let's change that and let's now hit slash projects slash two. There we go. We have data for slash project slash [00:03:30] two, but let's actually try to hit something that doesn't exist. For example, slash 20. This is not the ID of a project that exists. And because we haven't really done anything to have a good exception handling process over on the server side ... We're going to do that of course, but we haven't done anything up until this point ... this is what happens. So, when we hit this URL, what are we expecting back?

Well, we're expecting a 404 [00:04:00] not found. That would be the correct http status code that the server should reply with when it's not able to find the project. However, what it does reply with is a 500 internal server error. That's not correct, so that is our next step here is to handle exceptions. All right, let's go back to the code, and let's start improving our exception or error handling. We are going to dedicate a full lesson to this later on, [00:04:30] so we certainly won't fix everything here, but let's still have a quick look at what the problems are.

One problem I already mentioned, that is the status code that is not actually following correct http semantics. Another problem, this time at the code level, is the fact that we are not proactively checking the result here. We're simply doing a get on the optional that the find by ID returns. That's not ideal, because that's going to throw [00:05:00] a null pointer exception, and we are allowing here for the possibility of that exception to just be thrown. It's way better to take control over that process. Well, let's do a simple fix here first. And again, we'll certainly discuss this further on.

That is a simple way to check the optional and throw a specific exception — [00:05:30] in this case, the response status exception with a not found — instead of just allowing the get to throw an exception on its own. Let's redeploy. Let's go back to the client and let's hit this again. And there we have it. We are now getting the 404 not found.

All right, so now that we're done with the read operation, let's do a write operation as well. That's going to include other aspects of Spring [00:06:00] MVC, so that's an important step here. Naturally, we're going to do a simple save, which is basically a creation of a new project. So, first of all, let's define this basic operation with no mappings. Now, we're going to map these to an http post. Let's also map the body of the http request to the project input here. [00:06:30] Let's delegate to the service. And we should be good to go.

Now, let's switch back to the client and let's send our post. All right, so let's open up our prepared post here, our prepared create operation, and let's first have a look at it. So, what we have here is a simple post request. This time, just hitting slash projects. [00:07:00] And if we have a look at the headers tab, we're sending the content type as application JSON. Of course, we're sending an actual project as the body. So. there we have it.

This is the whole post. Let's send and let's have a look at what happens. All right, so we're getting back the 200 OK. And our create operation, of course, works as expected. There we have it, our very first simple controller. All right, I [00:07:30] hope you're excited. See you in the next lesson.