Document the API with Swagger

00:00: Welcome to the first module of Course Seven: Documenting the API with Swagger. There are a few options when it comes to documenting a REST API, but Swagger is certainly the most popular, so we are going to explore this one. The very first thing that we need to discuss is why we need to document REST APIs in the first place. A very hard line view on REST APIs is that they do not need to be documented, so we're going to briefly look at that aspect, and then we're going to look at the practical state of the market. Next, we are gonna see what solutions are available for REST API documentation. As I mentioned, Swagger is one of them and it is the most popular, but it is far from the only option. We're then going to discuss exactly what Swagger is. And of course, we're going to set it up within our Spring-based API, and we're going to go through the full implementation.

00:52: Alright. Before going on to the technical aspects of this module, let's have a quick look at the available solutions to document a REST API. Swagger is obviously the first as well as the dominant solution in the market, just in terms of overall adoption. And of course, since Swagger is in the title of this module, you know that we're going to explore it, but let's also have a quick look at the alternatives. RAML is definitely an interesting solution, and it is seeing some good adoption and some good coverage out there. It's a more declarative solution than Swagger, it has a pretty complex semantic, and overall, it really does a good job documenting an API. And there are many other solutions to this problem. API Blueprint and Apiary are just a couple of them, which I have not personally used, but definitely the market is primarily Swagger with RAML as a second good alternative.

01:40: Okay. We're going to start our Swagger implementation by adding the Maven dependency, of course, in our web application pom here. And it is important to understand that we are going for the Swagger 2.0 specification. We are not using Swagger 1.0, which had a different dependency. We are, of course, using the latest official version, which is Swagger 2.0. There we go. This is the dependency, and you'll notice a few things. First of all, the project that brings Swagger support in the Spring ecosystem is called Springfox, and this is the semi-official Swagger implementation that will actually help us bring Swagger support into our API. You'll also notice that as I was just saying, we are targeting Swagger 2.0 here. And finally, it's also important to understand that this particular dependency, it's only the core Swagger support. It does not include the UI aspect of Swagger, only the core support that enables us to generate Swagger data according to the Swagger spec for our Spring API.

02:37: So next, we are going to enable Swagger in our configuration, and then we're gonna have a look at how the data that gets generated by Swagger, how that looks. Alright, now comes the fun part which is enabling Swagger within our configuration here, and we're going to do that in the web configuration just because it's the natural place where Swagger can live. So the first thing we're going to do is we're going to enable Swagger via an enable annotation here. So pretty straightforward, we have this EnableSwagger 2.0 annotation which we can just use in the standard enable style that Spring provides. The next step is to start configuring Swagger. Enabling it is one thing, but now we need to tune the Swagger behavior to fit what we really want out of the documentation. So let's do that by defining a main bean that will control the entire Swagger configuration.

[pause]

03:40: So here we go. We defined the main configuration here. This is the docket bean. This is going to be our entry point in controlling the entire configuration of Swagger. So let's start actually creating that configuration now.

03:56: And there we go. This is, of course, a very simple start to what will be our main Swagger configure.

[pause]

04:13: We've now started to use the builder pattern in order to further configure this docket, and the first thing we did is that we ensure that Swagger will pick up all of our request handlers that define our Spring API. We did that with this predicate requesthandlerselectors.any(). And as you can see, there are a few options here. We want ini because we only have the API in this particular application. But if your application is bundling more than just an API, then you can certainly go in and select a more restrictive predicate here and basically pinpoint the API that you need covered by Swagger.

04:54: We are doing the exact same thing for our paths because basically, all of them need to be covered by the Swagger docs. Next, we're specifying to Swagger what the base path of our servlet, the servlet that is serving our API, what that is, and in our case, it's /API.

05:17: And now, we're doing some extra configuration. This is a very straightforward way to configure global substitutions of models, which in this case basically means that we want local dates to simply be replaced by strings.

05:34: And finally, and this is definitely an interesting configuration aspect, is that we really want Swagger to not document response entities. What we actually want is the data within the response entity. So the response entity wraps a certain kind of data and uses that certain kind of data as a generic parameter. And so we don't want the response the entity, which is a Spring-specific type of data. We don't want that showing up in our Swagger documentation. We want the actual data type that gets wrapped in this response entity. That's what we really want to show up in our main documentation. And of course in that way the documentation remains highly relevant as well as Spring agnostic. We don't really want to Spring specific artefacts such as the response entity, we don't want that in our main Swagger documentation. So there we have it. This is basically the main Swagger documentation, and of course this is something that we can keep tuning and keep configuring, but this is certainly a very good base to build on because Swagger really doesn't need that much configuration. And now with Swagger enabled and configured, let's actually see what kind of data it generates once we started up. Okay, so the server is running and we now have the URL of the API docs, the data that Swagger now generates to document our API. So let's have a look at that.

06:53: Okay. So first thing is that we're getting the "200 okay" back and we can see that the content type is application JSON. And there we go. This is the Swagger format. Now this is a pretty printed version, it actually looks like this. So not very pretty but let's have a look at this one again. And we can immediately see how Swagger works, and when in the next step we're going to enable Swagger UI, that UI application is going to display this data in a much more user-friendly way. So we can now see our paths, we can see our controllers, the potential responses for those operations, so it's pretty well documented, it essentially goes over everything that you might want to know about the API. But of course you really do need the UI to really make sense of this data. So that's gonna be our next step, enabling the UI.

07:47: Okay. So let's now add the Springfox UI dependency in here. Pretty straightforward, it's just Springfox Swagger UI, and now we have what we need in order to restart the server and see the UI. We do need to add some minor extra configuration in terms of serving the static resources of this UI. So let's now continue using the web-config since most of the Swagger configuration lives here. Let's continue using this class and let's make sure that we essentially configured Spring well in order to start serving the UI. The very first thing you'll notice here is that this is a predefined method. So we are yet again leveraging the configurer adaptor pattern that the Spring configurations usually provide. We are using here the WebMvcConfigurerAdapter and that's what is defining our add resources handlers method here. And so what are we configuring here? We add the Swagger UI as a Maven dependency, but now we need to tell Spring where the Swagger UI artefacts are really located so that it can serve them out of the MVC layer. And there we have it.

08:55: This is a very simple way to add Swagger UI into our application without having to manually add the static UI resources ourselves. So pretty straightforward. Let's now restart the server and let's finally start seeing how the Swagger UI makes sense of the data that we saw previously. And there we have it. We're now consuming the Swagger UI out of our own web application/API/Swagger UI HTML. So with that minor configuration we were able to now serve the full Swagger UI as well as tie our own API documentation here in the Swagger UI. And we can now explore the UI here and we can basically see all of our operations, we can see the mappings here, we can see the types of operations, the HTTP verbs, we can see the names here. Of course these can all be configured but even without extra-configuration the UI here looks really good, really helpful, and actually really close to something that you might go into production with. Alright, so what did we learned in this module?

10:01: Well, first we learned why we really need to document an API, and why the theoretical approach of "API is not needing documentation" is not something that holds water in the current state of the industry. We then had a quick look at the state of the eco-
system and where Swagger fits in that eco-system. And finally, we did a full implementation of Swagger, documenting our REST API, and using Springfox. Alright, hope you're excited. See you in the next one.