Options Pattern

To make the application really robust and flexible at the same time, there is a pattern called the Options Pattern. Okay, let's jump right in, and let's change our current implementation in order to support the options pattern. In order to be able to use the options pattern at all, we need to register it as a service, or rather, map the part of the configuration to the specific strongly typed model. To do that, we need to call the configure method of the IServiceCollection, use the ApiConfiguration as the type, and then provide the section of the configuration we want… We can do that by using the GetSection method as we did before. After that, we can remove the ApiConfiguration initialization and configuration binding, and we can easily access our service by using the ServiceProvider's GetRequiredService method and then typing the IOptions<ApiConfiguration> as the service type. Then, we can access the values by using the Value property of our options object and get the BaseAddress. At the first glance, not much has changed, but we've effectively removed the manual initialization of the configuration, and we've given that job to the dependency injection container. We've also registered the service this once, and we can now use it throughout the application without too much trouble. Let's go to the Product repository and see how we can inject our configuration now. First things first, we need to change the IConfiguration to the IOptions<ApiConfiguration> in our constructor. This means we don't actually inject the whole configuration every time we need to read a specific part of it, but just the section we need. We can inject as many sections as we need. Now we can see that the compiler is not happy about this change because it expects IConfiguration. That said, let's assign the _apiConfiguration field with the "Value" property as we did in the program class. Great. That's all we need to change! If we check out the file upload method again, we'll see that we can access the base address as we did earlier, and everything is nice and strongly typed. There's only one place we need to use the string to map the configuration, and thus we've made sure that errors are far less likely to happen. We've also utilized the in-built dependency injection mechanisms to achieve this and thus decoupled our dependencies as much as possible. Now, we can just return the environment variable to Development. As we've already mentioned, we won't talk about IOptionsSnapshot and IOptionsMonitor interfaces, but these interfaces are really useful when building server-side applications. We encourage you to look them up by following the links below the video.