Summary

* We started out with using PUT for full updates. It's important to remember that all fields should be updated as a result of a PUT request. Fields that aren't passed through in the request should be set to their default values. A successful PUT request warrants a 200 Ok or 204 No Content status code. And PUT is not safe as the resource changes, but it is idempotent. Executing the request multiple times should result in the same result.
* These days, PUT is being replaced by PATCH. That allows for partial updates, removing the overhead PUT requests can have. The body of a PATCH request is defined by the JSON Patch standard. It describes how to pass through a list of operations that have to be applied to the resource. Successful PATCH request also warrants 200 Ok or a 204 No Content status code, just like PUT. PATCH is very powerful, and it's neither safe nor idempotent.
* We also learned about the special case, upserting, or creating a resource with an update request, be it PUT or PATCH. This can be done when the consumer of the API is allowed to create a resource URI. In regards to validation, it's important to remember that validation on update may be different from when you're creating a resource. To minimize code duplication, we can work with abstract-based classes that contain common validation rules and mark properties on that class as virtual for those properties that need additional rules applied. Also, when patching, it's the patched DTO that needs to be validated, both on update and upsert. For that, we use a TryValidateModel method. We can then return a ValidationProblem from a controller action that uses the same InvalidModelStateResponseFactory we configured before by overriding the ValidationProblem method and getting that factory instance from the RequestServices. That's it for updating resources.