Reflections Daumantas Indreika aka Justice3000 reflections.pdf

I reflections-filen under rubkriken "Säkerhet" har du motiverat dina säkerhetsåtgärder i applikation. Svara på hur har du kontrollerat och begränsat vilken information som finns tillgänglig vid anrop till API:et? samt hur inloggning och utloggning fungerar i klientgränsnittet.

Security

As per instructions, as a best practice I've used already created AspNetCore.Identity framework. This way my application gets great advantage of being maintained by thousands of people and therefore I skip the necessities of getting my head over every single possible scenario.

In my case I've created a register and an authenticate methods, these can be called anonymously. Register method is very simple, I take the registration model with email, name, password and query if this should be admin/user account. Then Create method is being called in my user_repo. There I hash and salt user's password, while adding him to desired role (I've used Policies for that matter) and then just return OK 200. The user yet cannot any resources on API.

Now I require authenticate API call. There user/admin is supposed to send two parameters in POST call body: email and password. Upon receiving mentioned prerequisites, server inspects if such user is found and if now again hashed password does match the stored hash from the registration. Green light here does fire up the JWT token creation process. Upon token creation, the server secret string is being used and claims are being generated. Here besides simple user id claim I do add a second one which describes a role. Now 1 hour expiration is added and everything is hashed into a string. I do return user OK 200 with body of user's id, role, email, name, last name, username and the token string.

This information may be used against user for less then 1 hour now, or until new authorize call is being made and new data is received. Every access point in the API (except the ones marked as anonymous) does require an authorization. This marking is made as an decoration attribute which instructs AspNetCore Identity that these resources may be accessed only by know users. Furthermore my added policies does take care of access for different *kind* of users. A simple decoration with attribute like `[Authorize(Policy = "studio")]` informs Identity that this specific point shall be accessed only by an authorized user and only if if that user belongs to "studio" role. Failure does provide an 401 unauthorized error. Upon successful call API does not return the original entity. I've used automapper with dedicated models to take care of that and if needed quickly hide or add any information necessary.

A design like this allows clear and flexible login process from WEB. First user just needs to send a body with required data to designated registration point, and upon success just make another call to authenticate point (with some parameters as email and password inside body) to receive a JWT token. This token is just needed to be sent with any other API call. It means that no password is being stored on the client side and that no password is easy to read on the server side either. "Logging out" on the client side is as easy as a removal of token. When stored token is being removed (in my case from localStorage) no API call can be made. Therefore localStorage.clear and a page reload is sufficient.

I reflections-filen under rubkriken "Klientgränsnitt" har du motiverat hur du tänkt kring användarbarheten i din klientapplikation utifrån ovanstående kravlista, vad var viktigast att få med? Anser du att gränssnittet är användarvänligt?

Client Interface

My client interface is horrendous and should be hidden as symbol of one of humanity’s failures. Because of my absolute lack of experience and low brain power it does not work as I wish it would. To begin with color palette, popping dialog boxes asking are you sure, smooth transitions, responsive buttons and so on. There is so much missing here. I’ve used only vanilla JS, absolutely no frameworks and simple HTML and CSS. I’ve personally found it very hard to write everything from the ground up, without any how to structure JS code example seen before (I would love to see at least a single project on Pluralsight on whatever techniques how to and where to build your code, flow fundamentals). Therefore all you can see is spaghetti code on top of some other spaghetti mess. There are so many mistakes there, but I’ve got so many insights now. While building this interface I’ve realized why the API was built the way I’ve built it. For front-end rich models are a big plus in my opinion (here I refer to received data), but everything else that is being sent back to server should be as small as possible and preferably easy to find. The most important things to get into WEB-API are authorization, registration and resources with rich descriptive data (preferably not nested into million nests), so that server calls could be smaller and easy to read. Client interface is very time consuming. Unfortunately my client side is not that user friendly because some of the actions have to be repeated over and over, like clicking button to open modal view with available films, to see movies. It if it is open it will not automatically update it’s contents.

I reflections-filen under rubkriken "REST" har du förklarat och motiverat hur du designat dina åtkomstpunker och resurser för att uppfylla kraven på Webb-API:et

I reflections-filen under rubkriken "REST" har du förklarat och motiverat hur du designat dina åtkomstpunker och resurser för att uppfylla kraven på Webb-API:et

REST

For simpler clarification let’s take access point for film. Let’s pretend that I’ve just done a GET *api*/v1/Films. The only thing server does care about is that you are in fact you. As long as your token is valid, there are no other prerequisites. Everything is absolutely stateless here, and API does not rely on the implementation of interface. They’re completely separated, therefore client just sends a request to retrieve or modify, and server responds. If request is valid, the answer provided to to inform that everything is 200 OK or 201OK in case of creation. The resources provided back in some requests are the data that user might want to see just in case it is incorrect. API call always returns the latest requested object and to see if there are any changes, a new call is required.

I reflections-filen under rubkriken "Implementation" jämför du och motiverar vilka interna modeller som finns och om/hur de skilljer sig från de synliga resuserna i Webb-API:et

Implementation

Resources at the API access point, are just mapped models of the real entities. This prevents unnecessary exposure and differs from the originals.

As an example my User.cs entity is a derivative from IdentityUser, that allows me to customize my users. Here my model does have fields like role, passwordSalt and passwordHash, but they are never seen in the registerModel that I use for registration. In fact when registering the only message user gets is that user has been created, meanwhile when authenticating another model is being created as an OK response which just spits out things like id and even role, but not the entity itself.

Another example could be Film.cs entity. It has many fields and even after mapping all of them are visible (except ID). You would think that it makes API vulnerable, but apparently no! Because almost every single resource that is sent is being mapped, you as an user can see only the imprint. Furthermore in my models I use some decorations which allows me pre-validate data that is being given to me. My film models also include a counter of how many films are in use and how many are left, but those cannot be manipulated by anyone since those have only getters, meanwhile the entity does have get and set methods (how otherwise server would set them). This whole mapping thing pretty much allows me to manipulate data between API and user, without exposing internals of any film specific IDs in my database context, or any other sensitive information(I may choose which).