Reflektion – Filmstudion - Johannes Jakobsson

REST

Vi kan börja med resurserna som är Film, Filmcopy, Filmstudio och User. Alla dessa behövs i min applikation för att uppfylla kraven. Dessa klasser har även var sitt interface som de uppfyller.

Alla repositories är namngivna precis som klasserna jag nämnde i ovan stycke men avslutas med -Repository. Dessa uppfyller även dom var sitt interface. I dessa repositories finns alla metoder som lyssnar efter controllerns mottagande CRUD anrop och utför dessa efter användarens val och sparas därefter i databasen. Jag försökte i början att anamma så mycket funktionalitet som möjligt till repositoryt. Men ju mer komplext det blev och ju längre tiden gick desto mindre följde jag det målet. I vissa åtkomstpunkter sker mer i controllern än vad som var tänkt, vilket jag är aningen missnöjd med. Mitt mål var att få en så "ren" och "tydlig" controller som möjligt. I slutändan fick funktionalitet och att hinna färdigt i tid gå före.

Jag har delat upp mina åtkomstpunkter på fyra olika controllers FilmsController, FilmStudiosController, MyStudioController och UsersController. I FilmsController och FilmStudiosController finns det några få publika åtkomstpunkter där användare som inte har registrerat eller autentiserat sig har tillgång till data, dessa är markerade med [AllowAnonymous]. UserControllern är helt öppen för att användare ska kunna registrera och autentisiera sig. Resten av åtkomstpunkterna är skyddade med hjälp av JWT-tokens och [Authorize] attribut högst upp i controllern. Användare som redan är registrerade kan autentisera sig och få tillgång till hela eller delar av API:et beroende på deras roll, detta görs genom åtkomstpunkten api/users/authenticate tillsammans med ett JSON-anrop där det ska finnas ett korrekt användarnamn och lösenord. Då skapas det en JWT-token som kopplas till denna användare och vidare för att komma åt låsta delar av API:et behöver denna JWT-token skickas med i anropets header "Authorization" för att få tillgång till data.

En User kan vara antingen en Admin eller en Filmstudio. Om en användare väljer att registrera sig som User/Admin så måste denna skicka med en bool IsAdmin: true i anropet för att en User ska registreras. En filmstudio ska inte skicka med en bool IsAdmin:true, den har andra krav och fält som behöver uppfyllas för att en användare ska bli registrerad med rollen filmstudio och för att en ny filmstudio ska registreras som tar plats i API:et. Det är alltså två separata åtkomstpunkter som skickar olika JSON-anrop för att olika användare ska kunna skapas.

För att skicka och ta emot olika former av information, alltså exempelvis gömma information beroende på roll, har jag främst arbetat med klasserna som ligger i mappen Resources. Jag har förstått det som att man kan namnge dessa väldigt olika beroende på preferens. Allt som befinner sig i Resources mappen är olika modeller/resurser som behövs för att "hemlig" data inte ska skickas till fel personer och för att forma en förfrågan/request efter preferens eller krav. Med hjälp av automapper mappar jag informationen mellan olika klasser för att värdet ska förflyttas mellan dessa klassers variabler och för att korrekta fält i JSON format ska komma till rätt användare beroende på autentiserings-roll. Det finns en mapp som heter Helpers. Där i finns alla skapade mappningar.

I början valde jag att följa Jason Watmores design av ett REST-API men ganska snabbt frångick jag hans sätt att bygga och började implementera egna grejer och utforska andra sätt att koda på. I hans exempel så använder han inga asynkroniska funktioner/metoder. Jag vet och förstår syftet med async-await, dock vet jag inte exakt när man "måste" implementera det och inte. Jag kollade igenom ett klipp med Shawn där han säger att man förr eller senare behöver lära sig att arbeta med det. I mitt projekt använder jag inte async-await någonstans vilket skrämmer mig lite såhär i efterhand

eftersom vi jobbar med data man ska hämta. Men API:et har ändå fungerat för mig hittills, bortsett från otaliga buggar och ej fullt färdiga delar. Async-await frågan är något jag behöver ta med mig och läsa på om vidare.

Implementation

De synliga resurserna antar jag är delar av mina resurser/modeller som ligger i Resources-mappen. I det näst sista stycket under rubriken REST har jag förklarat lite mer om mina Resources och hur jag använder dessa. Jag har skapat ganska många olika resurser för att försöka få det så tydligt som möjligt och att inget onödigt ska skickas mellan anropen och för att rätt användare ska ta del av rätt information. Varje resurs uppfyller ett interface utifrån kravlistan så att rätt information skickas i JSON-strängen vid varje anrop. Jag har förstått det som att "ursprungsklasserna" alltså de "interna modellerna" helst ska hållas interna och man ska skapa modeller/resurser som skickar ut vald data/information. I huvudsak för att man ska kunna lägga till variabler och information i de interna modellerna utan att de exempelvis automatiskt blir exponerade i API:t. I mitt projekt har jag försökt anamma det tänket så mycket som möjligt, jag har det dock inte på alla åtkomstpunkter för att i vissa fall skickar jag hela objektet ändå och anser att det inte behöver mappas utan blir med "rörigt" än vad det tillför. I mappen Resource har de resurserna som är ett svar från api:t "Response" i namnet och en "requests/förfrågan" innehåller inte ordet "Response".

Säkerhet

I huvudsak sker Authentication och Authorization med hjälp av JWT-Bearer-token. Först måste man självklart skapa en användare för att ens kunna autentisera sig och få tillgång till en token. Vissa controllers har ett [Authorize] attribut som gör att man måste autentisera sig med en giltig token för att få tillstånd att komma åt åtkomstpunkten. Vid varje åtkomstpunkt i dessa controllers är det första programmet gör att hämta ut användarnamnet från medskickad token och jämföra gentemot användare i databasen för att se om användaren finns. När användaren är lokaliserad kan man enkelt med hjälp av if-satser begränsa vilken information som ska skickas tillbaka beroende på exempelvis roll eller bool-attribut som IsAdmin.

I klientgränssnittet är det första som görs att kontrollera om en token är sparad i localstorage. Är den sparad kommer användaren direkt att loggas in. En token är dock bara giltig i ett dygn. Finns det ingen token sparad så behöver användaren navigera till inloggningsmenyn. Väl där behöver ett användarnamn och lösenord skrivas in. Därefter skickas ett POST anrop till api/users/authenticate, kommer svaret tillbaka med statuskod 200 och att användaren är en Filmstudio så sparas token, användarnamn och studioId i localstorage, samma värden sparas också till lokala variabler för att göra det lättare med de andra funktionerna i programmet. När allt detta är gjort skapas en välkomstsida tillsammans med en logga ut knapp. För att logga ut klickar man på knappen. Man förflyttas då tillbaka till logga in sidan och allt i localstorage samt de lokala variablerna töms på innehåll för att ny användare ska kunna logga in utan problem.

Repo: https://github.com/jajo21/Filmstudion