Dokumentation av projekt i DAT076

Oskar Jönefors
Jesper Olsson

Klient

Klienten är skriven med ramverket Angular 1. Klienten är uppdelad i olika moduler/komponenter, där varje komponent ansvarar för en viss del av layouten/funktionaliteten i applikationen. Navigationen sker med hjälp av biblioteket ‘ui.router’ och dess $stateProvider. Nedan beskrivs de olika modulerna och deras ansvarsområden kortfattat.

home

State: ‘home’ - Ansvarar för att hämta och visa de 20 mest populära filmerna, de 20 senast släppta filmerna och de 10 senast skrivna recensionerna. All kommunikation till vårt REST-API gällande home sker genom ‘home.factory’

lists

State: ‘lists’ - Ansvarar för att hämta och visa upp alla listor som finns i databasen och även att hämta och visa upp alla listor som har skapats av personer som användaren följer.
Direktiv: ‘list’ - Ansvarar för att visa upp en specifik lista, samt att ta bort den specifika listan och även att öppna modalen där användaren kan redigera sin lista.
Modal: ‘add-list-modal’ - Modal där användaren kan skapa och redigera listor genom att söka på filmer, och sedan drag and droppa dem till sin lista.

All kommunikation med vårt REST-API för denna komponent sker genom ‘list.factory’

login-register

Direktiv: ‘login-register’ - Direktiv som ansvarar för att visa ‘Sign in’ eller Sign out’ längst till höger på menyn. Vid klick på ‘Sign in’ skall direktivet öppna login-register modalen och vid sign out skall den logga ut användaren, ta bort kakan med användarens token och rensa användarinformationen som finns i ‘user.factory’.
Modal: ‘login-register’ - Modalen där man kan skapa och logga in en användare.

All kommunikation med vårt REST-API gällande login sker via ‘login-factory’ och all kommunikation gällande registrering sker via ‘register.factory’

menu

State: ‘menu’ - Detta är föräldertillståndet till alla andra tillstånd i applikationen. Navigationen till de olika komponenterna sker mestadels genom detta tillstånd.

search

Direktiv: ‘search’ - Detta direktiv ansvarar för att visa upp sökfältet, samt att byta tillstånd beroende på vad användaren vill söka på (movies eller users). Den senaste söksträngen och de senaste sökresultaten sparas i ‘search.factory’.

movies

State: ‘movies’ - Detta tillstånd ansvarar för att visa upp två saker: Om användaren inte har sökt på någon film så skall användaren kunna bläddra bland populära filmer i en så kallad ‘infinite scroll’ där tillståndet hela tiden laddar in nya filmer när användaren närmar sig ‘botten’ av webbläsarfönstret. Om användaren har sökt på någon film så skall tillståndet visa upp de 20 bästa sökresultaten.
‘movie-detailed’ - Detta tillstånd ansvarar för att visa upp detaljerad information om en film**,** såsom en kort beskrivning av handlingen, vilka genres som filmen tillhör och de senaste recensionerna av filmen. I detta tillstånd kan man även lägga till en recension på den givna filmen.
Direktiv: ‘movie’ - Detta direktiv ansvarar för att visa upp en specifik film i en grid av filmer och används på flera ställen i applikationen, bland annat i home, movies och lists. Direktivet kan ta olika posterstorlekar som parameter.

All kommunikation med vårt REST-API gällande filmer sker genom ‘movie.factory’, medan kommunikationen med TMDb’s API sker via ‘tmdb.factory’.

profile

State: ‘profile’ - Detta tillstånd ansvarar för att visa upp och redigera profilen för den användare som är inloggad. Här kan användaren redigera sina uppgifter, ta bort och redigera sina recensioner, ta bort och redigera sina listor samt avfölja användare som personen följer. All kommunikation till vårt REST-API gällande profiluppgifter och användare som personen följer sker via ‘user.factory’, medan information gällande användarens skapade listor sker via ‘list.factory’ och information gällande användarens skapade recensioner sker via ‘review.factory’.

reviews

State: ‘reviews’ - Detta tillstånd ansvarar för att hämta och visa upp alla recensioner som finns i databasen, samt att hämta och visa upp alla recensioner som har skrivits av personer som den inloggade användaren följer.
Direktiv: ‘review’ - Direktiv som ansvarar för att visa upp en recension. Ansvarar även för att redigera och ta bort samma recension. Direktivet används på flera ställen; ‘home’, ‘reviews’, ‘profile’.’ other-user’.
‘add-review’ - Direktiv som ansvarar för att lägga till en recension på en film. Direktivet används i tillståndet ‘movie-detailed’ och består av knappen ‘Add review’, som vid klick

ändras och blir till en textruta där användaren kan skriva sin recension, fem stjärnor där användaren kan fylla i sitt betyg samt knapparna ‘Save’ och ‘Cancel’.

All kommunikation med vårt REST-API gällande recensioner sker via ‘review.factory’.

users

State: ‘users’ - Tillstånd som ansvarar för att hämta och visa alla användare i databasen. Det ansvar också för sökningen på användare.
‘other-user’ - Tillstånd som ansvarar för att hämta och visa information om en specifik användare, som inte är den inloggade användaren (som tillståndet profile ansvarar för). Här visas användarens användarnamn, profilbeskrivning samt recensioner som den användaren har skrivit och listor som den här skapat.
Direktiv: ‘follow-user’ - Direktiv som ansvarar för att visa upp ett användarnamn och en knapp som antingen säger ‘Follow’ eller ‘Unfollow’, beroende på om den inloggade användaren redan följer denna användare. Detta direktiv används i ‘Users’ tillståndet för att visa vilka användare som finns registrerade, samt ge den inloggade användaren möjlighet att följa och avfölja denna specifika användare.

Server

Serverapplikationen är skriven i Node.js. För att hantera endpoints och initial middleware såsom tokenvalidering och inputsanitation används Express.js. Användarna autentiseras med JSON Web Tokens.
MongoDB används som databas, för sin flexibilitet och goda integration med Node.js. För att modellera databasobjekten används Mongoose, vilket ger en mer intuitiv och lättläst kod än vad som är fallet då man interagerar direkt med databasen istället.

Endpoints

När Express tagit emot en request och utfört eventuell tokenvalidering och inputsanitering går den vidare till en endpoint där det kontrolleras om requesten har alla de parametrar som krävs. Därefter slussas den vidare till relevant middlewarefunktion.

Följande endpoints finns tillgängliga:

Metoder	**URL**	**Beskrivning**
POST	/login	Inloggning
POST	/register	Användarregistrering
GET, PUT	/profile	Hämta/uppdatera profiltext
GET, PUT, DELETE	/following	Se följda/följ/avfölj användare

GET, POST	/lists	Hämta/skapa filmlistor
GET,PUT,DELETE	/lists/ID	Hämta/uppdatera/radera specifik filmlista
GET	/lists/following	Hämta filmlistor av följda användare
GET	/movies/ID	Hämta snittbetyg och recensioner för film.
GET, POST	/movies/ID/reviews	Hämta/skapa recension(er) för film
GET	/reviews	Hämta ett valfritt antal recensioner från samtliga användare eller följda användare, sorterade på röster eller datum om så önskas.
PUT, DELETE	/reviews/ID	Rösta på/Radera recension.
GET	/users	Hämta samtliga användare, eller sök bland användare om söksträng skickas med.
GET	/users/USERNAME	Hämta specifik användare.
GET	/users/USERNAME/profile	Hämta användarens profiltext.
GET	/users/USERNAME/reviews	Hämta användarens recensioner.
GET	/users/USERNAME/lists	Hämta användarens filmlistor.
PUT	/users/me	Uppdatera den egna användarens information, såsom email, lösenord och användarnamn.

Middleware

Här ligger all egentlig funktionalitet. Medan tidigare stadier mest har bestått av validering är det här som kommunikationen med databasen sker. Logiken är uppdelad i följande moduler:

movie-api: Kommunikation med TMDB.
movie-lists: Skapa/Redigera/Hämta/Radera filmlistor.
movies: Hämta filminformation
profiles: Skapa/Redigera/Hämta profiltexter.
reviews: Skapa/Redigera/Hämta/Rösta på/Radera recensioner
tokens: Verifiera och signera JSON Web Tokens
users: Registrera/logga in/redigera/hämta/följ/avfölj användare

Databas

Databasmodeller

Movie

Innefattar den relevanta informationen för presentation av en film i en lista: titel, TMDB-id, releasedatum och länk till filmposter. Första gången en film efterfrågas så gör backend ett kall till TMDB för att hämta denna information och spara ner i databasen. Följande gånger hämtas informationen direkt ifrån backend. Databasmodellen har även en tidsstämpel för när informationen senast hämtades. Varje gång filmen hämtas från backend kollas samtidigt hur aktuell informationen i databasen är. Om den är äldre än ett dygn görs ett kall till TMDB:s API för att hämta hem den senaste informationen och uppdatera denna i den lokala databasen.

MovieList

En lista av filmer med en författare, listtitel och (valfri) beskrivning.

Profile

I nuläget enbart en profiltext för användaren. Skulle kunna uppdateras i framtiden för att innehålla mer information.

Review

En recension för en film. Har en författare, betyg, (valfri) recensionstext, samt en samling upp- och nedröster som ger användare möjlighet att tycka till om en recension.

User

Användarinfo såsom användarnamn, email, lösenord samt vilka användare som följs. Lösenordet hashas med bcrypt för god säkerhet.

Tester

För testerna används ramverket Mocha samt biblioteken Supertest och Should.

Flöde för att skriva en recension på en film

Låt oss säga att användaren vill skriva en recension på en film. Användaren är redan inloggad och står just nu på /home. För att skriva en recension på en film finns flera alternativ. Användaren kan bläddra bland filmer på /movies, eller söka på en film. Låt oss säga att användaren vill söka på en film:

1. *Klient:* Användaren väljer ‘Movies’ i dropdownen bredvid sökfältet.
2. *Klient:* Användaren börjar skriva in titeln på filmen i sökfältet.
3. *Klient:* Direktivet ‘search-directive’ har en lyssnare på söksträngen som aktiveras varje gång söksträngen ändras. Om söktypen är inställd på ‘Movies’ och söksträngen inte är tom så gör direktivet ett kall till funktionen searchMovie som finns i ‘movie-factory’
4. *Klient:* Funktionen searchMovie tar söksträngen som input och skickar sedan detta vidare till funktionen executeTMDbRequest som finns i ‘tmdb.factory’
5. *Klient:* Funktionen executeTMDbRequest anropar TMDbs API med söksträngen som en query-parameter och får tillbaka ett Promise med sökresultat. Detta Promise returneras sedan vidare till searchMovie, som i sin tur returneras vidare till ‘search-directive’, där funktionen anropades initialt.
6. *Klient:* ‘search-directive’ resolvar sedan Promiset som returnerats och hämtar ut resultatet och sätter det till objektet searchResult i ‘search.factory’.
7. *Klient:* Tillståndet ‘Movies’ har en lyssnare som kollar om objektet searchResult i ‘search.factory’ har ändrats och när det har gjort det och söktypen har varit ‘Movies’ så visar det upp sökresultatet för användaren i form av en grid av ‘movie-directive’.
8. *Klient:* Användaren klickar på den film i resultatet som den vill skriva en recension om.
9. *Klient:* Tillståndet ändras nu till ‘movie-detailed’ och detaljerad information om filmen visas.
10. *Klient:* Användaren klickar på knappen ‘Add review’ som finns på sidan.
11. *Klient:* Användaren skriver sin recension och väljer sitt betyg i fälten som nu har dykt upp och klickar på ‘Save’.
12. *Klient:* Funktionen addReview, som definieras i direktivet ‘add-review’ kallas på. Den funktionen kallar i sin tur på funktionen createReview som finns i ‘review.factory’. Denna funktion tar recensionen, betyget och filmens id som parametrar.
13. *Klient:* Funktionen createReview skickar i sin tur ett http-request till backend endpointen /movies/:movieId/reviews och returnerar ett Promise med resultatet.
14. *Server:* Express.JS tar emot requesten i backend och skickar den genom middlewaren express-mongo-sanitize som kontrollerar så inte requesten innehåller ogiltiga tecken såsom $, som skulle kunna användas i MongoDB injection-attacker.
15. *Server:* Requesten går sedan vidare till middlewarefunktionen tokenVerfication som undersöker huruvida en sessiontoken skickades med requesten. Om så är fallet så valideras och avkodas denna. Om token är ogiltig eller utgången returneras ett fel. I token finns ett användar-id lagrat, och ett databaskall görs för att hämta användaren med detta id, vilken sedan bifogas till requesten innan den går vidare till nästa steg.
16. *Server:* Requesten går vidare till endpointen POST /movies/:movieId/reviews i routes/movies.js. Först undersöks om en användare finns bifogad i requesten. Om så inte är fallet så returneras ett fel i responsen. Annars kontrolleras att requesten har antingen ett betyg eller en recensionstext, och att dessa i så fall har giltiga värden.
17. *Server:* Funktionen reviews.postReview i /middleware/reviews anropas med betyg, recensionstext, användare och tmdb-id för filmen, vilket hämtas från URL:en.
18. *Server:* postReview anropar i sin tur funktionen movies.getMovie i /middleware/movies med det givna tmdb-id:t.

19. *Server*: getMovie söker i den lokala databasen efter en film med matchande id. Om ingen hittas så anropas funktionen movie-api.getMovie som hämtar filminformationen från TMDB:s API, och sparar ner denna i lokala databasen. Oavsett väg så returneras filmen i ett Promise.
20. *Server*: reviews.postReview fortsätter med att kolla i databasen om användaren redan har skrivit en recension för den angivna filmen. Om så är fallet så ersätts den gamla informationen med den nya. I annat fall skapas en ny recension. Slutligen sparas recensionen till databasen och returneras i ett Promise...
21. *Server*: ...återigen till endpointen i /routes/movies, som med recensionen instansierar
22. *Server*: ...en PublicReview, vilket är en extern representation av recensionen. Denna typ av representationer används genomgående för alla responser, och skapas genom att ta ett databasobjekt och enbart kopiera den information som ska exponeras. Detta för att inte av misstag exempelvis råka returnera en användare från databasen komplett med lösenordshash eller annan känslig information. PublicReview tar även mottagaren som en parameter, för att kunna skräddarsy den externa representationen efter denna. Exempelvis finns det möjlighet att rösta på recensioner, och då är det önskvärt att kunna visa för mottagaren hur denne har röstat.
23. *Server*: Då den externa recensionen har skapats returneras den i bodyn för http-responsen.
24. *Klient:* Funktionen addReview i ‘add-review.directive’ resolvar det promise som returneras av http-requestet till servern och lägger till recensionen i sin förälder-container (I detta fall är det ‘movie-detailed’).
25. *Klient:* Användaren kan nu se att sin recension har lagts till på filmen!

Klient: Kan tillägga att varje gång tillståndet förändras så kallas en funktion i app.js som verifierar att användaren fortfarande är inloggad och bestämmer sedan hur applikationen skall fortsätta. Också varje gång ett http-request skickas så fångar app.js upp detta och lägger till en ‘authorization’ header som innehåller användarens token, såvida inte requestet går till TMDs API. Då läggs ingen header på.