# movielister

## Grupp 4

### DAT076 - Web-applikationer

### HT2014


Magnus Rising
920311-2934
rising@student.chalmers.se

Vickie Dam
900122-4220
vickie@student.chalmers.se

Vidar Åsberg
930929-1590
vidara@student.chalmers.se

Alexander Ekstrand
920102-4552
aleeks@student.chalmers.se

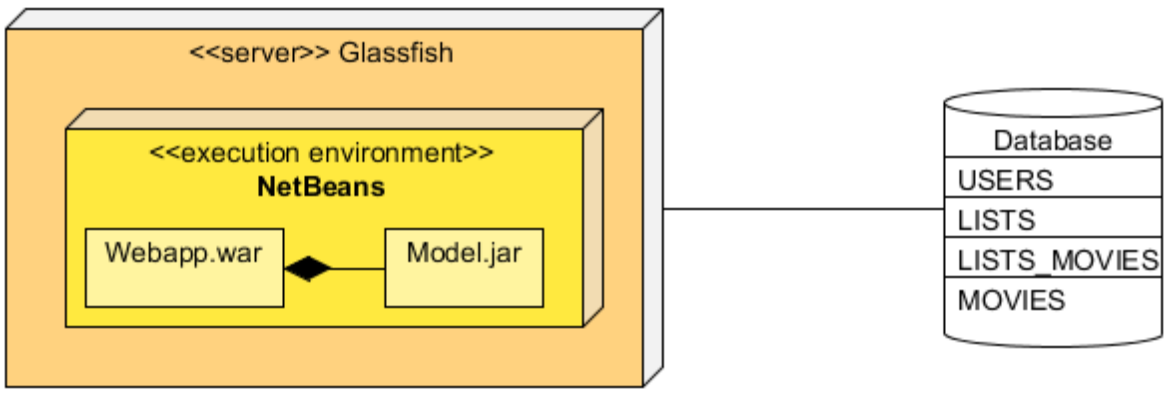
## Beskrivning av applikationen

Movielister är en tjänst för att upptäcka filmer genom topplistor samt göra egna filmlistor som man kan dela med sig av. Tjänsten kan användas utan ett konto då man kan söka på filmer för att se information om dom samt upptäcka nya filmer genom andras delade listor. Om man skapar ett konto får man möjligheten att skapa egna listor för att dela med sig av eller bara för sin egen skull, till exempel en lista med filmer som andra har rekommenderat och som man vill se i framtiden.

## Fungerande use cases

- Söka efter filmer
- Se information om en film
- Se publika listor
- Registrering
- Inloggning/Utloggning
- Skapa egna listor
- Ta bort egna listor
- Ändra namn på egna listor
- Ändra synlighet för egna listor
- Lägga till en film i en egen lista
- Ta bort en film ur en egen lista

## Enkel representation av systemet



## Design och paketstruktur

Vi har valt ett service-based approach där vi använder ramverket AngularJS på klienten. Applikationen är uppdelad i två Maven-projekt, **model** och **webapp**.

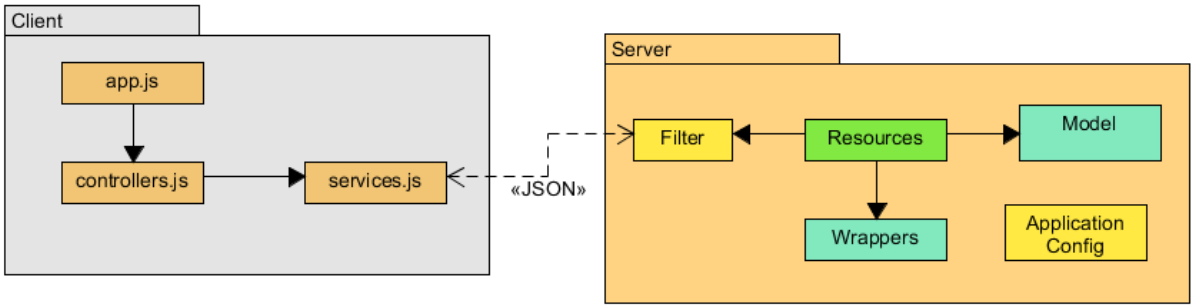
– **model** (*se bifogat UML-diagram)*

- **core** Innehåller entiteter, så som Movie, MovieList och User
- **dao** Innehåller klasser för läsning och skrivning till persistenslagret
- **persistence** Innehåller abstrakta klasser för persistenslagret

– **webapp**

- **auth** Servlets som sköter inloggningsflödet för OAuth
- **filter** Filter som skyddar resurser samt URL-manipulering
- **resource** Innehåller REST-gränssnitten

## Kommunikationen mellan klient och server




## REST API

| | | |
|---|---|---|
| GET | /lists | Hämtar alla publika listor |
| GET | /lists/{id} | Hämtar en publik lista |
| GET | /users/{user}/lists | Hämtar en användares alla listor |
| POST | /users/{user}/lists | Skapar en ny lista för användaren |
| GET | /users/{user}/lists/{id} | Hämtar en användares lista |
| POST | /users/{user}/lists/{id} | Lägger till en ny film i listan |
| PUT | /users/{user}/lists/{id} | Uppdaterar listan (namn, synlighet) |
| DELETE | /users/{user}/lists/{id} | Tar bort listan |
| DELETE | /users/{user}/lists/{id}/{movie} | Tar bort en film ur listan |
| GET | /session | Används för att hämta användarnamnet om sidan laddas om |

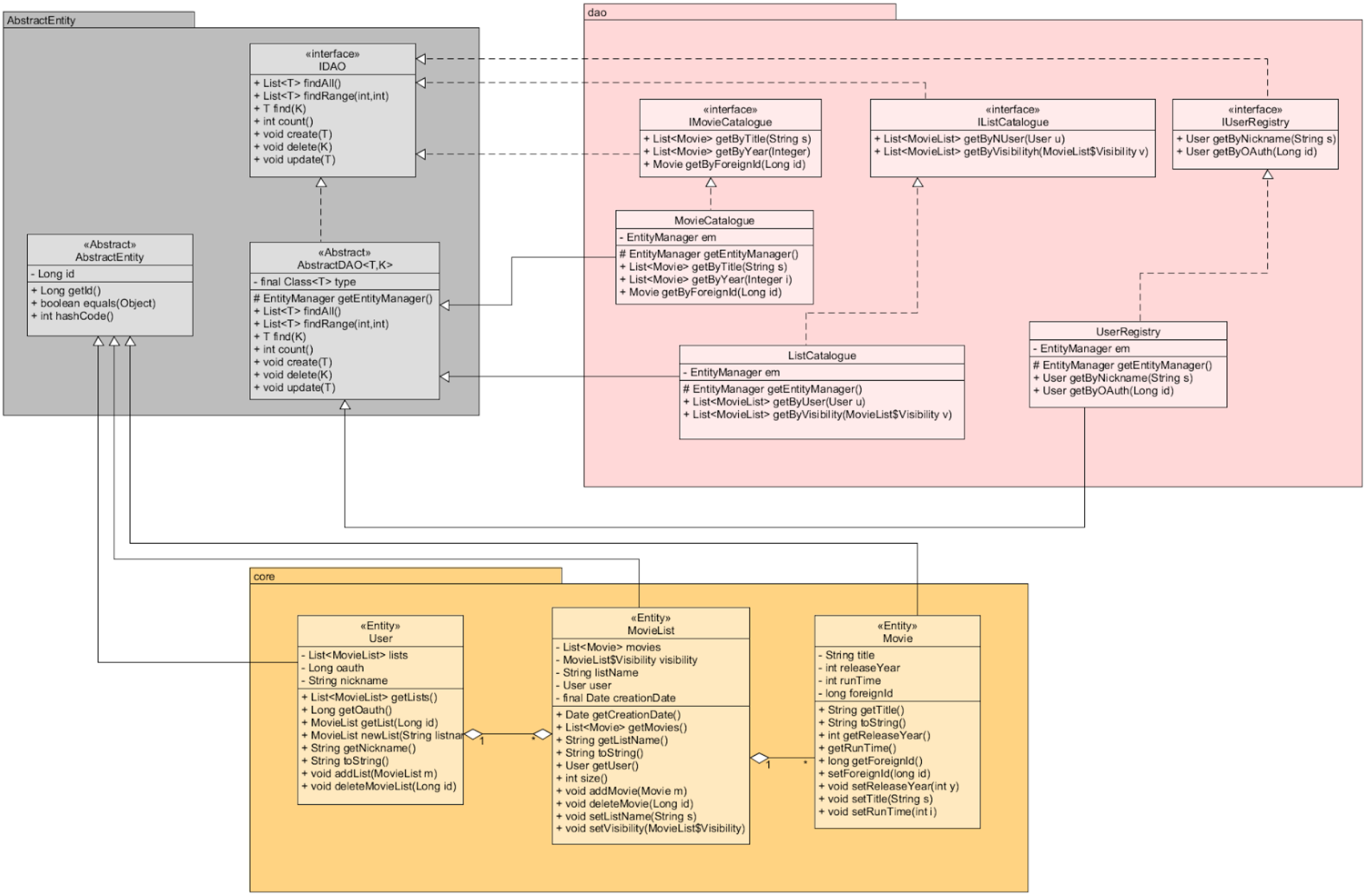
## Kommentarer till implementationen

Modellen är ett separat Maven-projekt som testas med Arquillian och ligger som en dependency i webbapplikationen. UML-diagram för modellen finns som bilaga.

Hanteringen av filmer i databasen är en uppbackning till den huvudsakliga källan, Rotten Tomatoes, då deras API inte tillåter den mängd förfrågningar som behövs. Således finns det inte en resursklass för filmer eftersom förfrågningar görs till tredje part.

OAuth används som autentisering och är implementerat med biblioteket SocialAuth från 3Pillar Global Labs. Första gången en användare loggar in skapas ett konto automatiskt och användarnamnet som används på tjänsten hämtas direkt från Twitter. Eftersom det unika användarnamnet används i URL:er finns resursklassen SessionResource som klienten använder för att ta reda på användarnamnet om sidan laddas om. Användarens privata data på tjänsten skyddas med ett resursfilter från javax.ws.rs som filtrerar resurser under `/users/{user}/` där `{user}` är det unika användarnamnet.

För att få korrekta URL:er utan nummertecken med ngRoute i AngularJS används UrlRewriteFilter från Tuckey.org. Eftersom konfigureringen av UrlRewriteFilter är begränsad i vissa avseenden ligger filterklassen RewriteFilter innan UrlRewriteFilter i kedjan för att exkludera REST-gränssnitten från omskrivning.


AbstractEntity
«interface»
IDAO
+ List<T> findAll()
+ List<T> findRange(int,int)
+ T find(K)
+ int count()
+ void create(T)
+ void delete(K)
+ void update(T)
«Abstract»
AbstractEntity
- Long id
+ Long getId()
+ boolean equals(Object)
+ int hashCode()
«Abstract»
AbstractDAO<T,K>
- final Class<T> type
# EntityManager getEntityManager()
+ List<T> findAll()
+ List<T> findRange(int,int)
+ T find(K)
+ int count()
+ void create(T)
+ void delete(K)
+ void update(T)
dao
«interface»
IMovieCatalogue
+ List<Movie> getByTitle(String s)
+ List<Movie> getByYear(Integer)
+ Movie getByForeignId(Long id)
«interface»
IListCatalogue
+ List<MovieList> getByNUser(User u)
+ List<MovieList> getByVisibilityh(MovieList$Visibility v)
«interface»
IUserRegistry
+ User getByNickname(String s)
+ User getByOAuth(Long id)
MovieCatalogue
- EntityManager em
# EntityManager getEntityManager()
+ List<Movie> getByTitle(String s)
+ List<Movie> getByYear(Integer i)
+ Movie getByForeignId(Long id)
ListCatalogue
- EntityManager em
# EntityManager getEntityManager()
+ List<MovieList> getByUser(User u)
+ List<MovieList> getByVisibility(MovieList$Visibility v)
UserRegistry
- EntityManager em
# EntityManager getEntityManager()
+ User getByNickname(String s)
+ User getByOAuth(Long id)
core
«Entity»
User
- List<MovieList> lists
- Long oauth
- String nickname
+ List<MovieList> getLists()
+ Long getOauth()
+ MovieList getList(Long id)
+ MovieList newList(String listnar
+ String getNickname()
+ String toString()
+ void addList(MovieList m)
+ void deleteMovieList(Long id)
«Entity»
MovieList
- List<Movie> movies
- MovieList$Visibility visibility
- String listName
- User user
- final Date creationDate
+ Date getCreationDate()
+ List<Movie> getMovies()
+ String getListName()
+ String toString()
+ User getUser()
+ int size()
+ void addMovie(Movie m)
+ void deleteMovie(Long id)
+ void setListName(String s)
+ void setVisibility(MovieList$Visibility)
«Entity»
Movie
- String title
- int releaseYear
- int runTime
- long foreignId
+ String getTitle()
+ String toString()
+ int getReleaseYear()
+ getRunTime()
+ long getForeignId()
+ setForeignId(long id)
+ void setReleaseYear(int y)
+ void setTitle(String s)
+ void setRunTime(int i)
1
*
1
*