These are the new requirements for the application:

- The application should be implemented using Spring Boot, Thymeleaf & MySQL
- The system should be able to create, read, update, delete, display all movies and a specified movie.
- A movie should have actors associated with it. The application should have CRUD functionality for actors and you should be able to add and remove them from a specified movie.
- A user should be able to search for movies on relevant content based on all the attributes that defines your movie.
- The application should be done using Git and should be handed in on GitHub with a link to your repository on Fronter.
- You should work with a master branch and each Use Case should be coded in a feature branch, and when done merged into the master
- You should document the analysis and design flow in Use Cases and by using the UML diagrams you worked with in 1. semester.
- You should make use of at least 3 GRASP design principles in your application.
- The Implemented Application should be deployed to your AWS infrastructure using Elastic Beanstalk
- The application should be able to connect and persist data to your RDS at AWS

Hand in and presentation

Your projects should be handed in on github, with a link on fronter latest sunday 28th October, and presented on 29th of October in class between 8:30 - 11.45 (Faisal and Claus will be there).

Actors: user

Use case 1: get movies
Main success scenario: a user enters ipadresse in browser and a site with movies turns up. Where all movies in the system will be push up.

Use case 2: create movie
Main success scenario: a user wants to add a new movie to the movies list. The user uses the button create and the system then takes the user to a new website where he/she will fill out the empty forms(title, year,genre, duration), after the info has been written the user push the button create and the system sends the user back to the movie list site with a new movie added.
Alternative flows:
Incase the requirements for the movie are not met, the program will display a “whitelabel error page”, or a “angiv et nummer” if the user writes a string instead of an int.

Use case 3: update movie
A user should be able to update the movie, including all the attributes about the movie. The user can then press enter and the selected movie will be updated with the changes.
Alternative flows:
Incase the requirements for the movie are not met, the program will display a “whitelabel error page”, or a “angiv et nummer” if the user writes a string instead of an int.

Use case 4: search movie
Main success scenario: the user searches on a movie title in the search bar, if the system finds a match to the search word, it will display that movie as a selectable option. The user can then select the movie and press “search” which will display the movie on the the display movie site.
Alternative flows:

Use case 5: read movie
A user should be able to select a specific movie, and then the program will display an information page about that movie, which includes all the attributes and actors.

Use case 6: Delete movie
A user should be able to delete a movie by selecting a delete icon on the far right side of the movies information panel in the display movie site. The system will then display a verification message saying “are you sure you want to delete this movie?” and the user can then press yes, which will delete the movie and update the site.
Alternative flows:
The user can decide to press no instead of yes to the verification message which will reload the site without any changes.

Brief Use case 4: get actors
Main success scenario: a user needs to see the list of all current actors. The user selects a button on top of the web page called actors

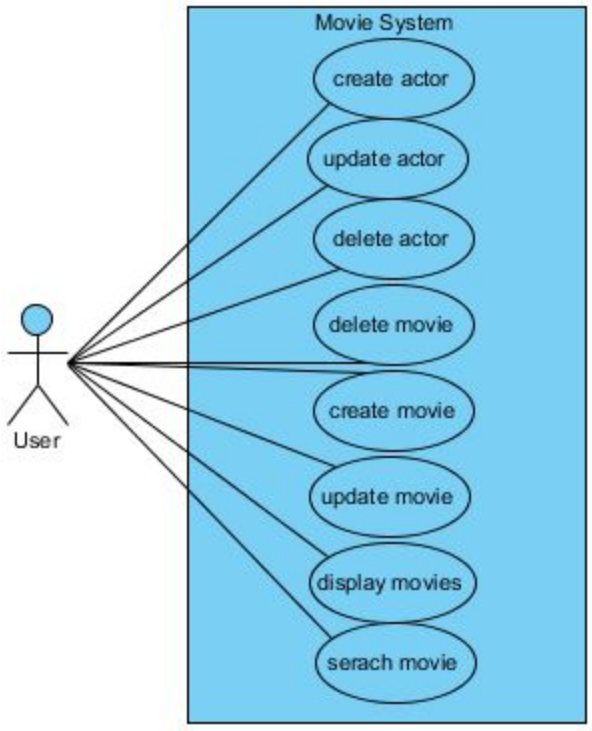
Casual use case 5: Create actor
Main success scenario: the user has to create a new actor into the system to attach to new movies, the user goes into the web page by pressing the the create actor button. The user then fills out the the empty forms(name), the system then register the data to the database.
Alternative flows:

1. **Update actor:** a user needs to change the of and actor he enters the web page where the actors is, and selects the actor the user wishes to change the system sends user to change the name and after the name have been changed. The system register the change.

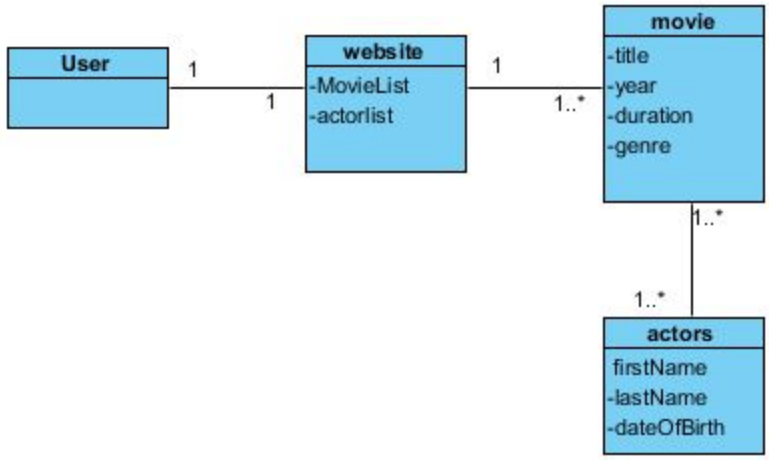
2. **Delete actor:** a user needs to delete a actor from the system, the user enters the display movie list and presses the delete movie button on the movie and system destroys the movie.

Noun list:
User
Movie
Title
Year
Genre
Duration
Button
Bar
System
Website
List

Use case diagram:

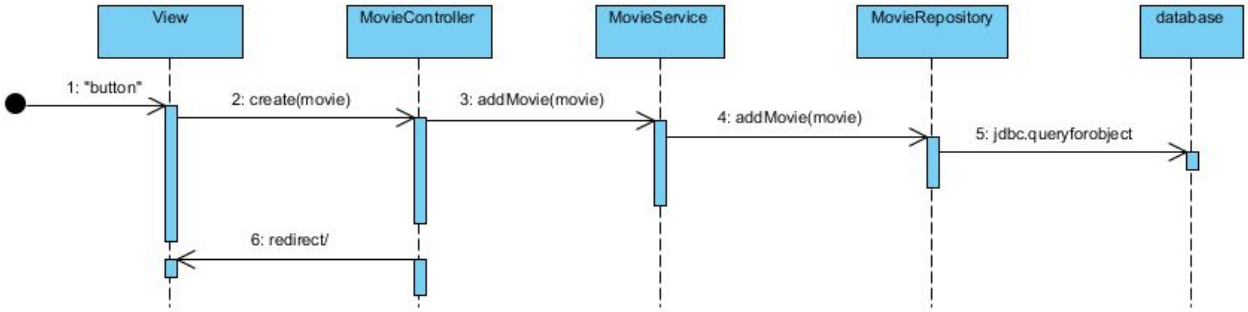


Domain model:

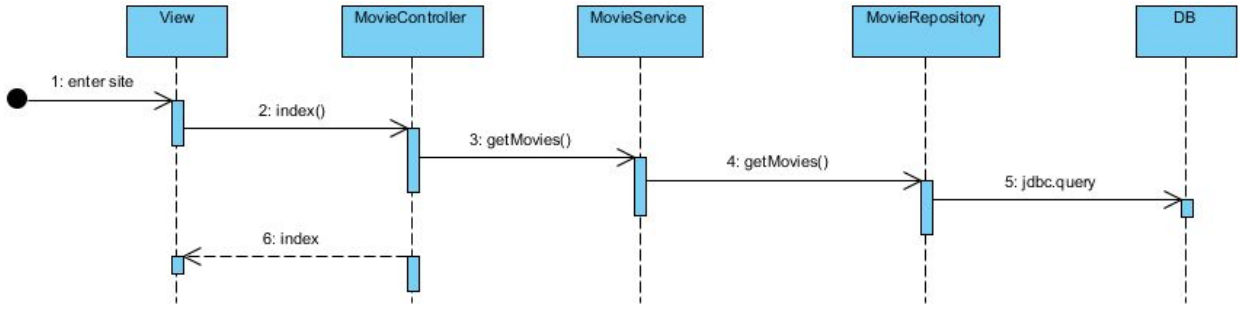


Sequence diagrams:

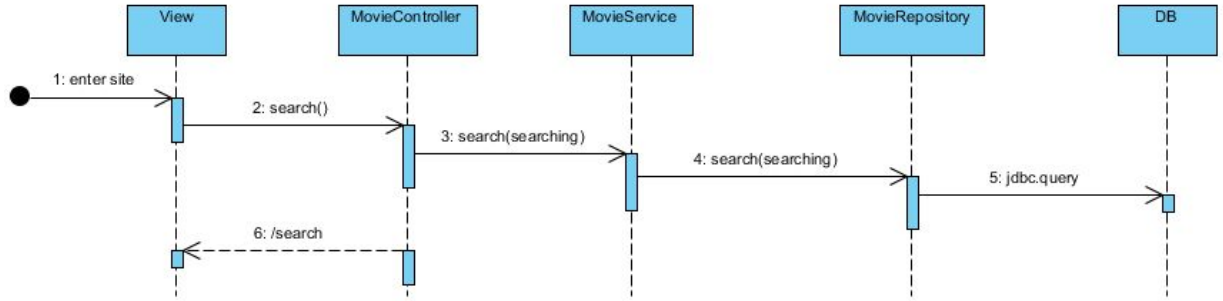
Sequence diagram: create movie



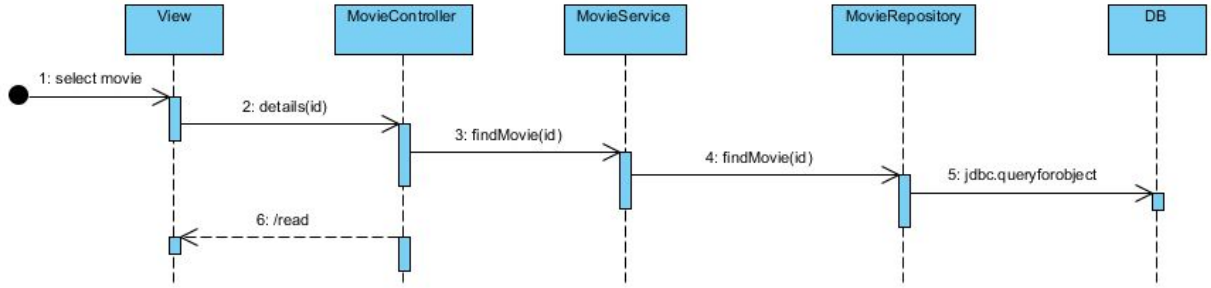
Sequence diagram: get movies



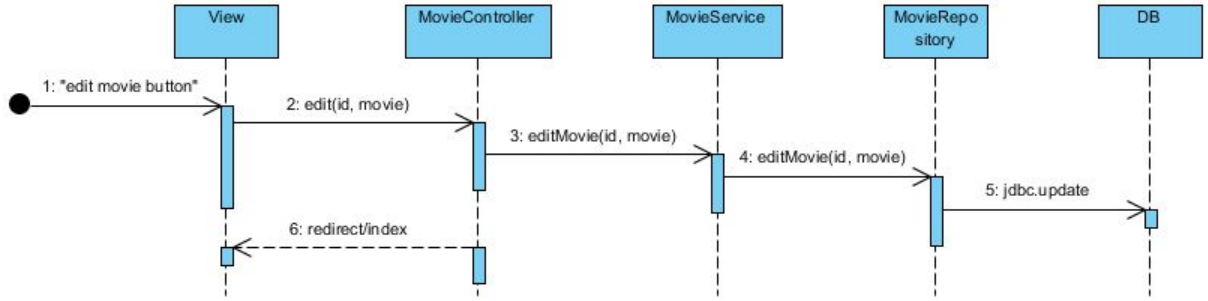
Sequence diagram 3: search movie



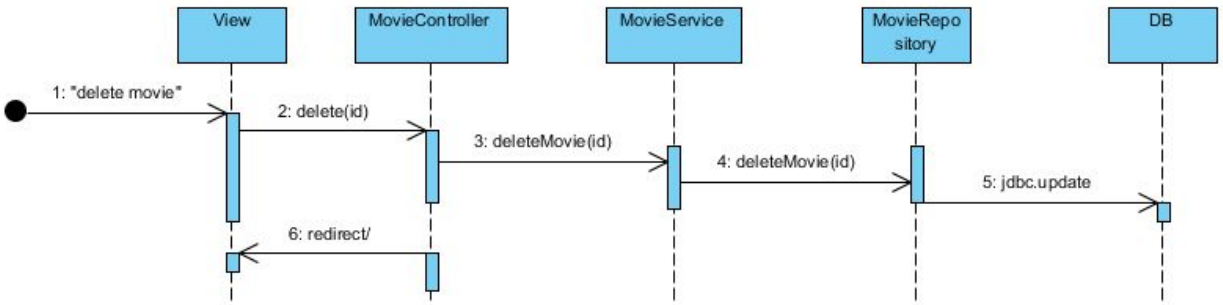
Sequence diagram 4: read movie



Sequence diagram 6: update movie



Sequence diagram 7: delete movie



Sequence diagram er stort set det samme men bare actor CRUD funktionlitet med actorrepo, service og kontroller.

Grasp:
Vi har andvendet Creator, low coupling, high cohesion som vores patterns til at lave vores class diagram.
Class diagram: