

# Forside

## Flixnet

3. semester projekt

Gruppe 1 (DMAB0917)

Andreas Larsen, Katrine Vindbæk og Mathias Lindberg

Dato: 17/12 2018

Antal anslag: 36562

Git navn + link: <https://github.com/linaqx/Projekt-3-Gruppe-6.git>

Git revisions nummer: 233

database navn: dmab0917\_1026423

# Indholdsfortegnelse

Flixnet	1
Indholdsfortegnelse	2
Introduktion	4
Problemformulering	4
System	5
Mock up	5
Domænemodel	8
Delkonklusion	9
Arkitektur overvejelser	9
Distribueret system	9
Server	10
Klienter	10
Database design	10
Database	11
Delkonklusion	11
Kommunikation og middleware	11
WCF	11
Protokoller	12
TCP/IP	13
OSI-model	13
Delkonklusion	14
Klienter	14
Web client	14
Native client	15
Delkonklusion	15
Håndtering af samtidigheds problemer	16
Transaktioner - ACID	16
Race conditions	17
Optimistisk samtidighed	18
Isolation levels	19
Locks	20
Delkonklusion	21
Sikkerhed	22
Sensitiv data	22

Attack vectors	22
SQL injection	22
Broken authentication and session management	23
XSS	23
Insecure direct object references	23
Security misconfiguration	24
Sensitive data exposure	24
Storing passwords	24
Hashing	24
Saltning	25
Delkonklusion	25
Test	26
Program guide	27
Web klient	28
Native klient	31
Konklusion	31
Litteraturliste	32
Bilag	33

# Introduktion

I Teknologi og programmering på 3. Semester er der blevet undervist i IT-sikkerhed, samtidighed, c#, web udvikling med mere og en ny platform Visual studio. For at vise hvad vi har lært opbygger gruppen et projekt og to rapporter, som gennemgår de vigtigste elementer fra undervisningen samt, hvilke udfordringer gruppen er stødt på undervejs og hvordan de er håndteret set igennem programmering og teknologi fagenes optik. For at opbygge et virkelighedsnært projekt har vi lavet en problemformulering, som er vores grundlag for programmet.

## Kode konvention

Gruppen har fulgt almene kode konventioner<sup>1</sup> herunder navngivningskonventioner med mere.

# Problemformulering

Hvordan kan man lave en distribueret platform som samler information om film og serier?

- *Hvordan kan man identificerer problemer, specielt samtidigheds problemer og udfordringer i udviklingen af en sådan platform?*
- *Hvilke løsninger, herunder særligt hvilken arbejdsmetode kunne man anvende for at imødekomme disse problemer og udfordringer?*
- *Hvordan kan man lave et distribueret system, som både repræsenteres gennem en app og en web applikation med samme funktioner og som er koblet op til samme database?*

---

<sup>1</sup> <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>

# System

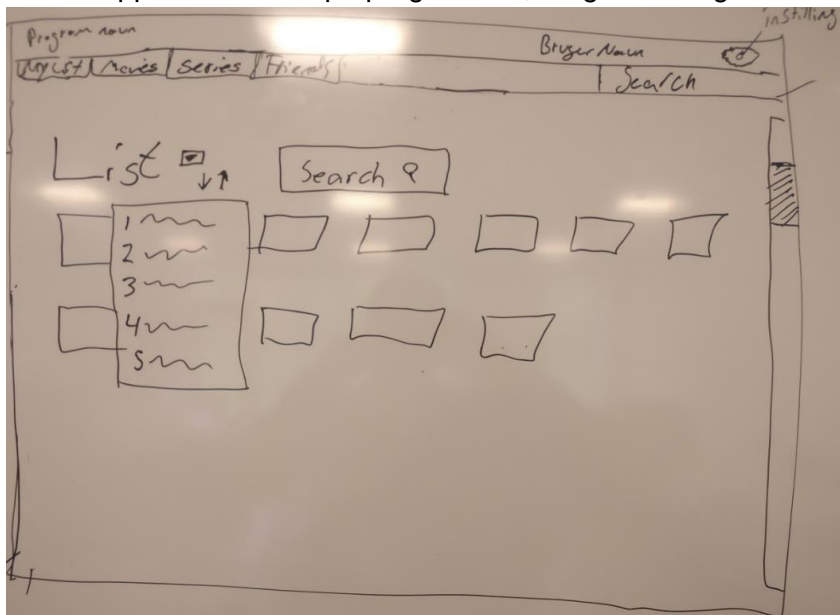
Her kan det ses, hvordan systemet er blevet udtænkt, hvordan brugeres user interface kommer til at se ud, hvilke klasser der er blevet udtænkt at have med i programmet, hvor der er blevet lavet en domænemodel og hvordan den relationelle model skal se ud.

## Mock up

De mock up det bliver vist er kun for brugere, så der bliver ikke vist nogle mock ups til, hvordan det bliver vist for en admin.

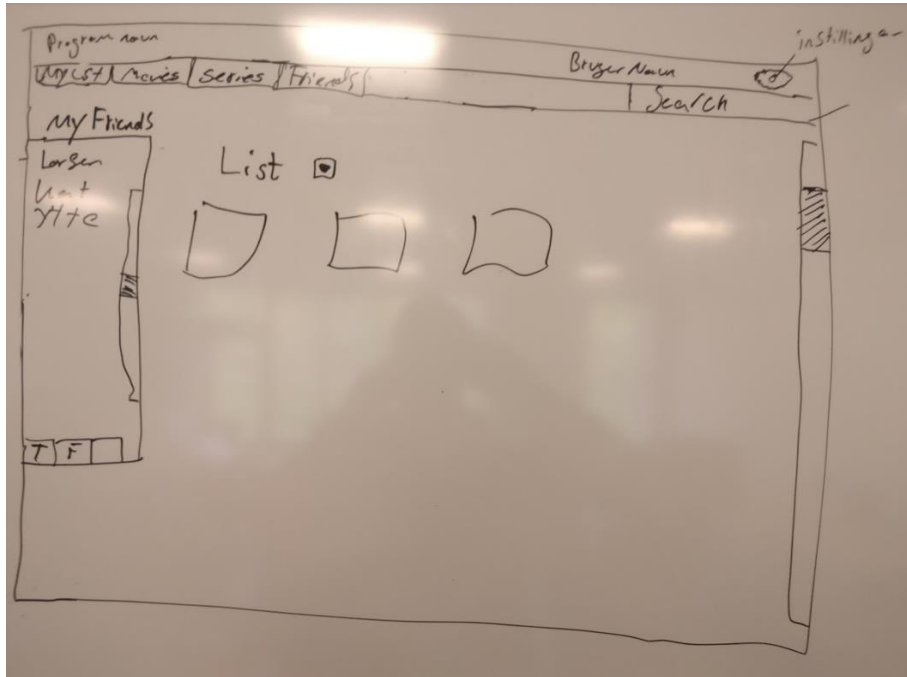
### Forside

På figur 1 kan forsiden ses. Det som bliver vist her er alle film og serier. Oppe i toppen kan det ses, at der kan vælges mellem forskellige sider og der kan søges på film og serier. Øverst oppe ses navnet på programmet, brugernavn og indstillinger.



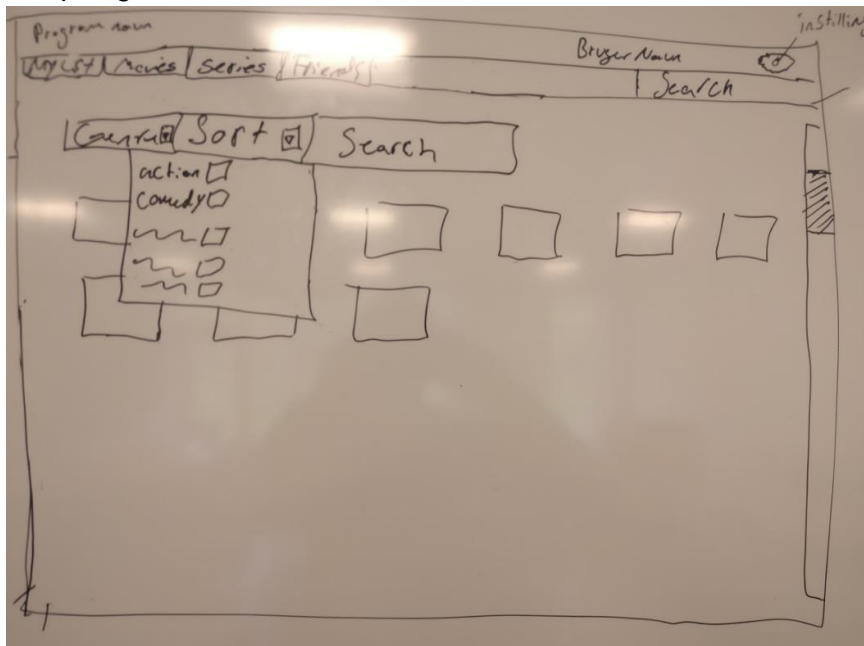
## My list

På figur 2 ses, hvordan my list er opbygget. Ude til venstre ses det, at en brugere kan have mere end en favoritliste, hvor brugeren kan vælge, hvilken liste de gerne vil se film og serier fra. Disse favoritlister har forskellige navne, hvilket gør det nemmere at genkende dem fra hinanden.



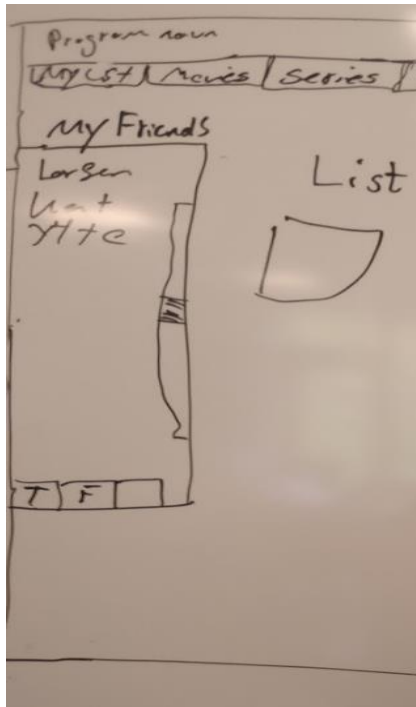
## Sortere ved genre

Her kan der søges efter film og serie på genre, hvilket vil gøre det nemmere for brugere at finde film og serier alt efter, hvad de har lyst til alt efter, hvad de har lyst til at se. Dette kan ses på figur 3.



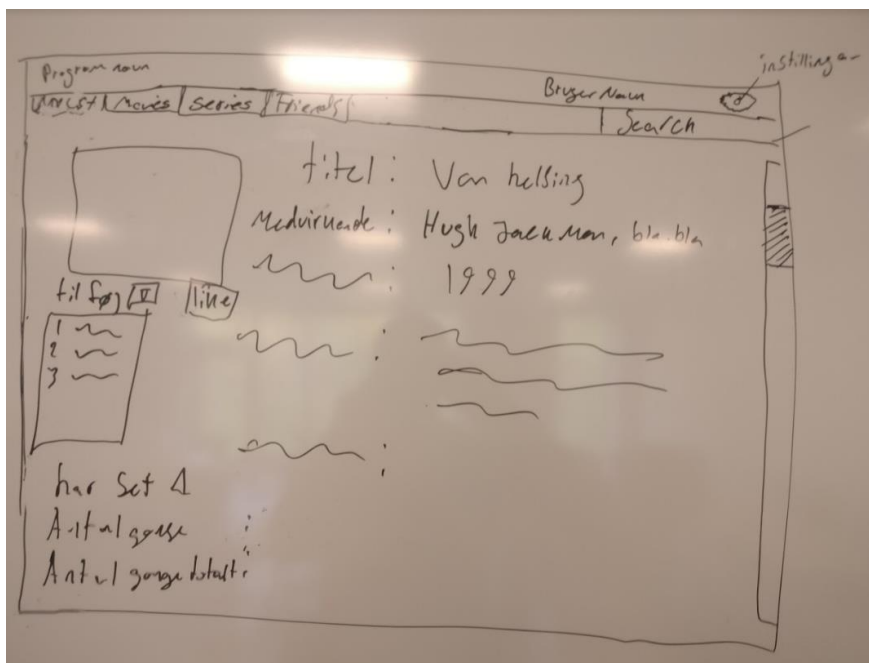
## My friends

På figur 4 ses, at en brugere kan have venner, hvilket gør det nemmere for brugeren at finde en ven, hvis disse to gerne vil have en fælles favoritliste.



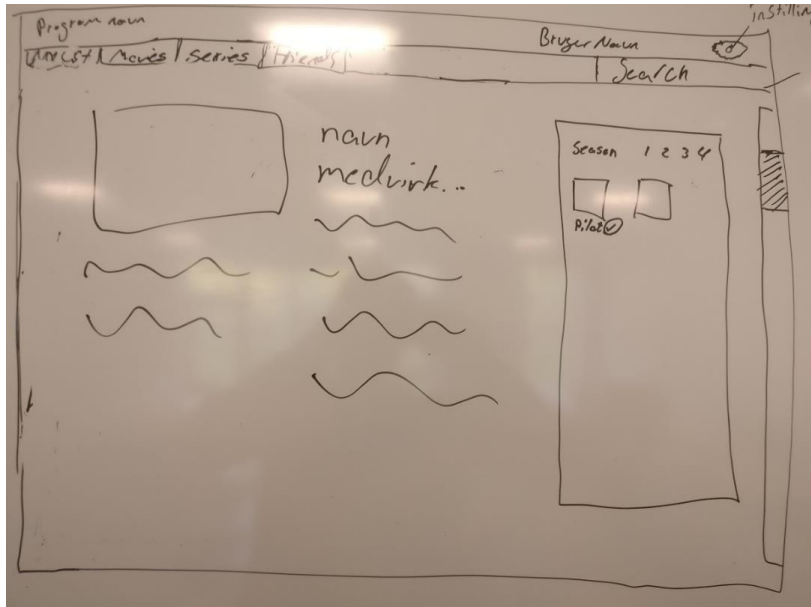
## Movie

På figur 5 vises det, hvordan alle informationerne omkring en film bliver vist, hvis en bruger har klikket på en film. Her vises der et billede af film. Filmens titel, hvem der har været med i den, hvilket årstal den blev udgivet og dens resume. Under billedet kan filmen tilføjes til en favoritliste, hvor brugeren selv kan vælge, hvilken af brugerens egne favoritlister filmen skal tilføjes til. Nederste i venstre side kan brugeren se om brugeren har set filmen, hvor mange gange brugeren har set den og hvor mange gange filmen er blevet set totalt af alle brugere.



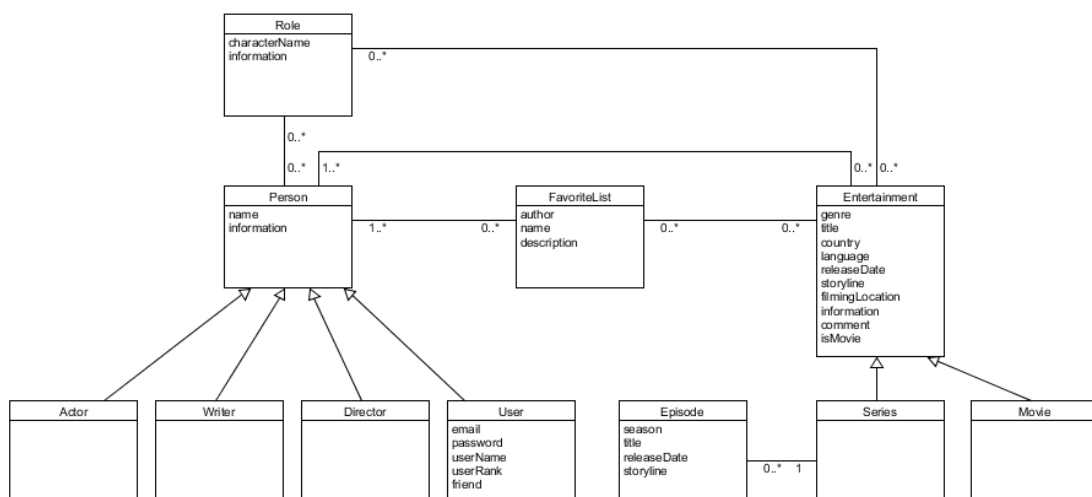
## Series

En series viser også et billede af serien, dens titel, medvirkende, årstal den første sæson blev udgivet til det årstal den sidste sæson blev udgivet og hvis den sidste sæson ikke er blevet udgivet så viser den også det og seriens resume. Her kan serien også blive tilføjes til en af brugerens favoritlister. Ude til højre kan der ses på en serie kan det også ses, hvor mange sæsoner der er og under hver sæson kan det ses, hvor mange episoder der er til hver sæson og episodens titel kan også ses. Her kan en brugere også følge med i hvilket episode brugeren er kommet til. Dette kan ses på figur 6.



## Domænemodel

Domænemodellen er blevet lavet, så der er en bedre oversigt over hvilke klasser og klassernes tilhørende attributter, der skal være med i systemet, desuden kan det også ses relationerne mellem de forskellige klasser. Domænemodellen kan ses på figur 7.





## Delkonklusion

Ved både at have lavet mock ups og domænemodel er det blevet nemmere at lave programmet og designet for user interface, da det er blevet udtænkt før at det blev lavet. Det har også hjulpet da det også kan ses visuelt, så det var nemmere at lave programmet, da det var et udgangspunkt.

## Arkitektur overvejelser

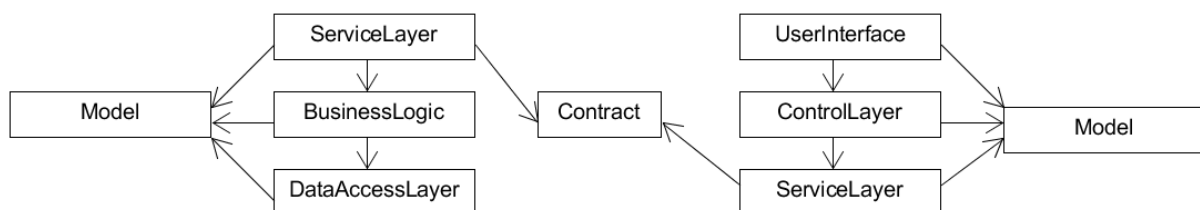
Arkitekturen der er blevet brugt er, hvor der er en server side og en klient side. Server siden indeholder servicelag, businesslogic, data access lag og et modellag. Klient siden indeholder user interface, kontrol lag, servicelag og et modellag.

### Server

Som det kan ses på figuren, så snakker alle lagene sammen med model laget. Data access er det lag som snakker sammen med databasen, så det er her den får alt information omkring film og serier. Kontrol laget er det lag som forbinder data access laget og servicelaget. Servicelaget på server siden er det lag som snakker sammen med servicelaget på klient side, hvilket vil sige at klienten får alt sin funktionalitet fra server siden.

### Klient

Ligesom på server siden så snakker alle lagene sammen med model laget. Servicelaget har som sagt forbindelse til servicelaget på server siden. Kontrol laget er den som forbinder servicelaget og user interfacet. User interfacet er så det som en bruger kan se når denne bruger, bruger programmet. Alt dette kan også ses på figur 8.



## Distribueret system

Det distribuerede system er en fællesbetegnelse for den struktur, som kan ses her Database -> Server -> Clients. I det distribuerede system er der 3 elementer.

### Database

Databasen indeholder alt den data, som skal benyttes af serveren og klienterne. Der findes mange forskellige database 'sprog' og platforme. I dette projekt er platformen SQL server management studio anvendt og sproget SQL. Fælles for alle databaser er, at de indeholder det data, der skal anvendes. Databasen er en individuel enhed i den forstand, at den i det distribuerede system kan udskiftes.

## Server

Serveren repræsenterer ikke en fysisk server, men et stykke software som skaber kontakten til databasen for derefter at bearbejde den data i et kontrol lag ved hjælp af et model lag. For til sidst at oprette en service, som indeholder de metoder, som udviklerne gerne vil anvende til deres klienter.

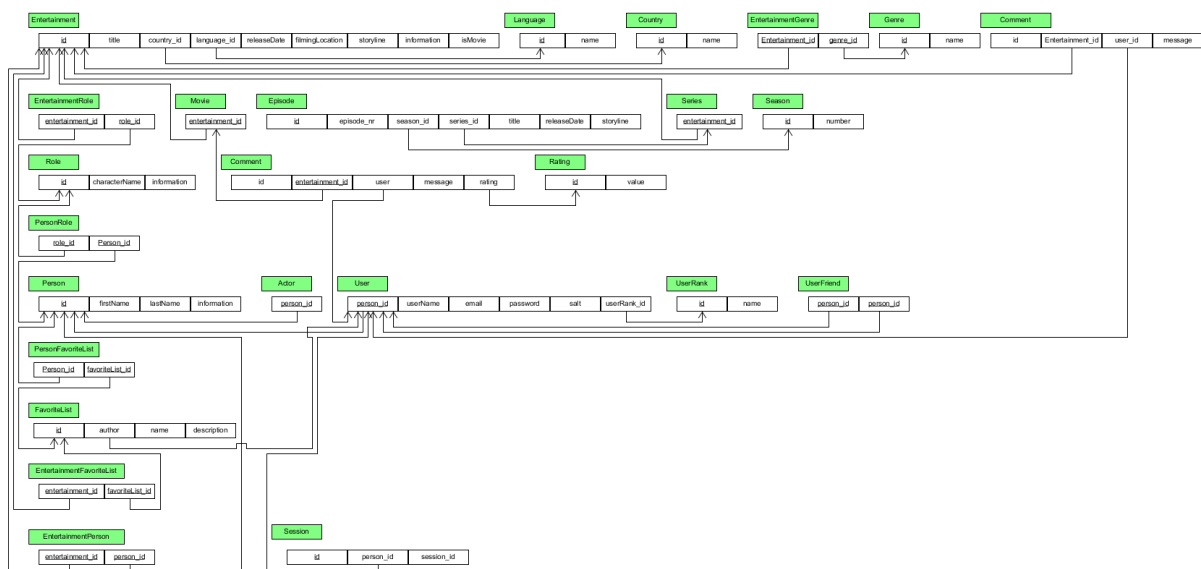
## Klienter

Der kan laves lige så mange klienter, som udviklings teamet ønsker, som opretter forbindelse til serveren. Klienterne opretter forbindelse ved hjælp af en service reference, som indeholder de metoder som fra serverens side er lagt ind i servicen. Services kan derfor også opdeles for at opnå bedre cohesion i systemet. Der er automatisk en lav coupling i et distribueret system imellem database, server og klienter. Klienterne får altså kontakt til service referencens metoder og kalder dem igennem deres eget kontrol lag for til sidst at vise det valgte data i et brugerinterface.

## Database design

Den relationelle model, som er blevet lavet ud fra domænemodellen, er blevet lavet så der er et bedre overblik over, hvordan databasen skal se ud.

Den relationelle model overholder normalform 1, 2 og 3, dog overholder den ikke boyce-codd, da de unikke attributter ikke er blevet valgt som primære nøgler. Denne kan ses på figur 9.



## Database

Databasen er blevet lavet ud fra den relationelle model, som bliver brugt til at gøre det lettere at opsætte databasen.

Det kan ses fra domænemodellen, at der er to steder, hvor der er klasser der arver fra en superklasse og det er 'Entertainment' og 'Person'. Her skal der foretages et valg om der skal laves pull up, pull down og table each. Der er blevet valgt at lave table each, så der ikke kommer null værdi eller redundant data. Der er også blevet valgt at bruge table each fordi det kunne tænkes, at der på et tidspunkt kommer flere attributter i klasserne 'Actor', 'Writer' og 'Director'. Ved valget af table each bliver det også nemmere at ændre på 'Actor', 'Writer' og 'Director', hvis dette skulle blive nødvendigt.

## Delkonklusion

Det distribuerede system er en ny arkitektur for gruppen. Det distribuerede system giver muligheden for at opdelle programmet i flere områder således at programmet bliver mere overskueligt derudover kan klienter udskiftes i denne arkitektur. I databasen har gruppen overholdt 3 normalform, databasen er præget af mange "mange-mange" tabeller, dette betyder dog bare at systemet indeholder mange lister.

## Kommunikation og middleware

I dette kapitel vil der blive beskrevet, hvordan middleware hjælper det distribuerede system til at kommunikere mellem de primære programmer. Det kan siges, at middleware er '-' i client-server. Det er programmer eller kommunikationsregler, som muliggør, at det distribuerede system kan opdeles i flere mindre elementer.

## WCF

WCF står for windows communication foundation og er en teknologi, som hjælper med at programmerer og hoste services på en windows platform. Den erstatter til dels gamle teknologier og samler flere funktioner. WCF sikrer, at man som udvikler ikke skal skrive den samme service for hver klient, der har brug for samme service.

WCF muliggør en Service orienteret arkitektur (SOA) det vil sige, at der er fokus på, at klienter nemt kan udskiftes fordi forbindelsen mellem klient og serverens service er svag. Det betyder at alt sikkerhed og kontakt til databasen skal håndteres gennem serveren ellers kan ondsindede programmører selv skrive en klient og derigennem påvirke databasen.

# Protokoller

Gruppens system er pr. 17.12-2018 hosted lokalt. Det vil sige, at der ingen netværksprotokoller anvendes ud over forbindelse til UCN's kraka database hvori SQL databasen ligger, men systemet er designet ud fra antagelsen om, at det i fremtid skal kunne tilgås gennem netværket. Derfor er det relevant at undersøge hvilke kommunikations protokoller der kan anvendes.<sup>2</sup>

Protokoller er et sæt af specifikationer eller regler til en bestemt type af kommunikation. Nedenstående kan ses to protokoller, som er relevant eller kan bliver det for gruppens system.

## HTTP

Hypertext transfer protocol<sup>3</sup> er for de fleste en browser standard og kan genkendes da de fleste hjemmesider starter med http:// eller https://. https tilføjer en enkryptering til kommunikation (s-> secure) originalt ved hjælp af SSL (secure sockets layer) og seneste TLS(Transport layer security) begge repræsenterer forskellige former for enkryptering. HTTP er et protokol lag bygget oven på TCP og kommunikerer fra en klient gennem en browser til en webserver. HTTP er "stateless", det betyder, at ligeså snart at en forbindelse mellem klient og server bliver droppet lige så snart request og respond er færdige. HTTP har tre primære request typer 'Get', 'Post' og 'Head', som kan hente og sende data fra serveren.

Projektet er planlagt som en hjemmeside og derfor kunne man i en videre udvidelse og arbejde med programmet hoste det fra en server således, at programmet ville anvende https. Det er vigtigt at få krypteringen med få at undgå en række angreb for eksempel man-in-the-middle.

---

<sup>2</sup> <https://www.lifewire.com/definition-of-protocol-network-817949>

<sup>3</sup> <https://www.lifewire.com/transmission-control-protocol-and-internet-protocol-816255>

## TCP/IP

Transmission control protocol/ internet protokol er i virkeligheden to separate protokoller, men bliver brugt sammen så ofte af TCP/IP er blevet en standard. TCP/IP er inddelt i fire funktionalitets lag datalink, netværk, transport og application. Når to terminaler anvender de samme protokoller kan terminalerne kommunikere.

- Datalink

Er sammenfattet af metoder og protokoller, som arbejder udelukkende gennem link, som er forbindelsen mellem to nodes (fysisk enhed som kan kommunikere over nettet).

- Netværk

Laget forbinder uafhængige netværk, som transporterer informations pakker over netværket.

- Transport

Transportlaget håndtere kommunikationen mellem hosts, og er ansvarlig for 'Reliability'.

- Application

Laget standardisere dataudveksling mellem applikationer.

TCP/IP er altså en omfattende række regler/protokoller, som ville blive anvendt i videreudvikling af programmet, hvis det bliver hostet for at kommunikere fra browser til web server

## OSI-model

Open systems interconnection reference model Indeholder 7 lag, som beskriver netværks kommunikation lagene er

1. Application
2. Presentation
3. Session
4. Transport
5. Network
6. Data link
7. Physical

Presentation, Session og Physical er de lag, hvor modellen adskiller sig fra TCP/IP. TCP/IP anerkende også disse lag men de repræsenteres gennem de andre lag.

- Physical

Det fysiske lag er de nodes, kabler, hubs med mere, som anvendes for at skabe kommunikationen over nettet.

- Session

Starter og stopper session som defineres som forbindelsen mellem terminaler.

- Presentation

Formaterer data'en så den modtagende terminal kan læse og forstå den, det er også i dette lag, hvor data kan krypteres eller dekrypteres.

## Delkonklusion

ASP.net er det primære middleware, som er anvendt til dette projekt. Det er et 'application framework' og tillader kommunikation mellem server og klient. Der benyttes ingen protokoller i projektet fordi web- og native klienterne ikke er tilgængelige fra world wide web.

## Klienter

I dette projekt vil de blive udviklet to klienter, disse to er en web client og en native client. Web klienten er lavet til brugere mens at native klienten er lavet til admins.

### Web client

For at udvikle webklienten er der blevet anvendt det arkitektur mønster, som hedder MVC. MVC står for model-view-controller. Model som er model-laget, der kan bygge objekter, View-laget som repræsenterer Html, CSS og Js, som er det, som brugeren kan se visuelt og kontroller-laget som er hjernen i systemet der bearbejder objekterne.

Derudover er der et Service lag som står for forbindelsen til serveren og dens funktionalitet. Gennem serveren er forbindelsen til databasen.

I Visual Studio er der en template for MVC<sup>4</sup> projektet, som gruppen har benyttet, det samme er gældende for authentication. Projektet er derefter blevet rettet til for at passe til gruppens vision for programmet og funktionaliteter er tilføjet.

MVC er planlagt til at inkludere forsvar mod XSS og forgerytokens

---

<sup>4</sup> <https://www.asp.net/mvc>

## Native client

For at opbygge gruppens native klient er window forms anvendt. Window forms er et integreret værktøj i visual studio, som hjælper med at opbygge et grafisk brugerinterface(GUI) med samme arkitektur som MVC. Et service lag som for forbindelse til serveren og derigennem databasen. De services som ligger i service referencen kaldes og bearbejdes gennem kontroller laget som endeligt kaldes fra GUI laget.

Native client er i dette projekt tiltænkt som et administrerende værktøj for programmet. Altså en klient som kun medlemmer af udviklingsteamet må tilgå. Den håndterer de fleste af de ændringer, som skal udføres til databasen for at gøre det sværere at påvirke databasen fra web klienten. Det er funktioner som create, update og delete der er tiltænkt native client. Web klienten har en enkelt update på favoritliste.

Klienterne har hver deres ansvarsområde om i en fremtidig udvikling eller udvidelse af systemet er de nemme at udskifte på grund af deres lave tilknytning til serveren og derigennem databasen.

## Delkonklusion

De to klienter som er en del af det distribuerede systemet skal varetage forskellige opgaver. Native klient er tænkt som et værktøj for administratorerne for web klienten. De kan redigere i databasen med hensyn til film og serier med mere. Web klient er tænkt som det interface brugerne interagerer med.

# Håndtering af samtidigheds problemer

I dette kapitel vil der blive beskrevet samtidighed og hvilken form for samtidighed der vil blive brugt i dette projekt.

## Transaktioner - ACID

Når man har et distribueret system, som har en database forbindelse, så arbejder man også med transaktioner. De fire elementer som gør sig gældende ved transaktioner er ACID, hvor det er vigtigt, at man overvejer systemets styrker og svagheder inden for de fire elementer. ACID står for atomar, konsistens, isolering og holdbarhed. De vil blive beskrevet herunder:

### 1. Atomar

- Gør at når en transaktion til databasen er i gang så udføres den fuldstændigt eller slet ikke.

### 2. Konsistens

- Under designet af databasen opsættes der nogle regler, som transaktionerne skal overholde for ellers er transaktionerne ikke konsistent.

### 3. Isolering

- Når en person er i gang med en transaktion så er der ingen der kan se den, mens man er i gang med transaktionen.

### 4. Holdbarhed

- Når transaktionen er gennemført, så er den gemt. Dette kan ikke ændres på også selv om at der sker en fejl som for eksempel en systemfejl.



## Race conditions

Race conditions kan opstå, når man har et distribueret system med mange brugere, som ønsker at lave ændringer i databasen på samme tid.

Se også tabel 1

Funktion	Konflikter
Read/Read	Nej
Read/Write	Ja
Write/Write	Ja

Et eksempel på dette i vores system kunne være når to eller flere brugere(admins) vil opdatere en film eller serie og gør det samtidig. Transaktionerne for dette kunne ses på tabel 2

Bruger A	Bruger B
Vil ændre i filmen "Van helsing"	Vil ændre i filmen "Van helsing"
Ændrer beskrivelse	Ændrer beskrivelse
Gemmer	Gemmer

Alt efter hvem som gemmer sidst, kommer der forskellige beskrivelser på filmen "Van Helsing" og den anden beskrivelse går tabt.

Der er to løsninger på dette problem, som kan implementeres enten med optimistisk samtidighed eller locks.

## Optimistisk samtidighed

Optimistisk samtidighed handler om sandsynligheden for, at der kommer en race condition, hvis denne er meget lav, vil man ikke implementere locks på de ressourcer/objekter, som skal benyttes.

Der implementeres i stedet for, et lavt isolation level i disse metoder, dette betyder, at der opbygges en metode ud fra dette:

- Read information
- Act on information
- Check if information is still the same
  - a. Complete
  - b. Fail → if “Check” fail
    - Abort method
    - Rollback changes
    - Try again

I dette system kunne optimistisk samtidighed for eksempel implementeres ved read/write på en film som skal opdateres. Her ville man forholde sig optimistisk til, at der ikke kommer en race condition, fordi handlingen (update movie) kun er tilladt af admin brugere, men fordi det stadig kan ske teoretisk, implementerer vi derfor ovenstående metode struktur.

Eksemplet på dette i systemet kunne være ved det samme som nævnt tidligere “update movie”, hvor to brugere(admin) vil opdatere “Van helsing”. Når ovenstående metode struktur er implementeret, vil der ikke kunne opstå en race condition, da transaktioner nu vil sådan her ud, se tabel 3

Bruger A	Bruger B
Read information “Van helsing”	Read information “Van helsing”
Makes changes	Makes changes
Check if information is still same as when method began	Check if information is still same as when method began
If fail: Abort the method Rollback changes Try again (fixed number of times)	If fail: Abort the method Rollback changes Try again (fixed number of times)
If complete: saves changes	If complete: saves changes

Hvis bruger A gemmer først, vil trinnet "Check if information..." for bruger B opfange, at informationen ikke længere er den samme. Hvis informationen bruger B begyndte med ikke længere er der den samme, skal der informeres om det igennem en fejlmeddelelse og må derfor starte rettelserne forfra. Derved undgås der race condition.

Grunden til "try again" trinnet skal være et særligt antal gange, er fordi flere brugere kan få et rollback try again samtidig. Sker dette får man en livelock, som betyder at mange brugere får et abort, rollback og try again. Når alle metoderne prøver igen samtidig, får man igen en abort, rollback, try again og dette kan fortsætte for evigt. En livelock kan løse sig selv, da der er en lille tilfældighed over hastigheden, som handlingerne sker i når de gentages, men dette kan tage lang tid og om muligt slet ikke ske. Hvis livelocken ikke skulle løse sig selv, har man sat et fixed antal gange den skal forsøge eller antal tid, før loopet stopper og man kan informere brugeren herom.

## Isolation levels

Som nævnt tidligere anvendes der et lavt isolation level ved optimistisk samtidighed.

Isolation levels er opdelt i 4 niveauer

1. **Read uncommitted**
  - Transaktioner har ikke et isolation level.
2. **Read committed**
  - Transaktion venter med at låse indtil, der ikke er nogen write lock på ressourcen.
3. **Repeatable read**
  - Transaktionen sætter en read lock på de ressourcer, som den tilgår og en write lock på de ressourcer den indsætter, opdaterer eller sletter.
4. **Serializable**
  - Låser hele tabellen, som transaktionen tilgår så, der ikke kan indsættes nye objekter så længe den arbejdes på.

Hvis der ikke implementeres isolation levels risikerer man 3 faktorer

1. **Dirty reads**
  - Når en transaktion læser data som ikke er committed altså gemt.
2. **Non repeatable reads**
  - Sker når en transaktion læser den samme information flere gange og den har ændret sig i mellemtiden.
3. **Phantoms**
  - Phantoms sker når en transaktion for eksempel `SELECT * FROM X` returnere 3 rækker af relevant data, men at der samtidig med at de rækker bliver bearbejdet, bliver der også oprettet en fjerde række som matcher søgekriteriet.

Hvis man vil undgå de ovenstående fænomener, skal man implementere et isolation level på niveau, se nedenstående tabel 4<sup>5</sup>

Transaction isolation level	Dirty reads	Nonrepeatable reads	Phantoms
Read uncommitted	X	X	X
Read committed	--	X	X
Repeatable read	--	--	X
Serializable	--	--	--

## Locks

Locks er en anden løsning på race conditions. Som betyder at man i read/write og write/write eksemplerne låser den ressource/objekt som skal behandles det vil sige, at der kun er en bruger af gangen, som kan behandle ressourcen/objektet. Man vil derfor aldrig kunne arbejde flere brugere på at opdatere en film og så videre.

Nogle metoder kræver to mutual afhængige ressourcer/objekter. Risikoen ved at benytte locks er derfor risikoen for deadlocks, som kan låse systemet. Hvis brugeren eksempelvis har

1. En fælles liste af favoritfilm.
2. En film (med et attribut der tæller op alt efter hvor mange favoritlister den er på).

og en metode som tilføjer filmen til fælles listen. Listen skal tælle en op på antallet af favoritlister den er på. Hvis to eller flere brugere benytter metoden samtidig på den samme film, vil både listen og filmen være låst for at undgå race condition. Hvis bruger A først får fat på fælleslisten og bruger B først får fat på filmen, vil metoden vente på den anden ressource for evigt og man har nu en deadlock.

---

<sup>5</sup><https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-2017>

For at en deadlock kan opstå skal der være fire kriterier til stede:

**1. Mutual exclusion**

- En ressource/objekt kan kun ejes af en process på et givent tidspunkt, dette sker ved låse og derfor giver det ikke mening at fjerne Mutual exclusion, da dette er hele meningen med at have locks i systemet.

**2. Hold and wait**

- Processen beholder ressourcen, mens den venter på en anden ressource som den skal bruge.

**3. No preemption**

- Når processen først har fat i ressourcen, som den skal bruge for at køre sin del kode, så kan den ikke give slip på sin ressource igen. Denne del kan også være svær at fjerne, da man helst ikke vil have afbrydelser i den del af koden.

**4. Circular wait**

- To eller flere processer venter i en cirkel som for eksempel i dining philosophers dilemma.

Deadlocks kan undgås ved at fjerne en af de fire kriterier. Løsningen i dette system, hvis der vælges at implementere locks, ville være deny hold and wait. Dette ville være den simpleste løsning at implementere for eksempel, at når en film skal tilføjes til fælles listen må den kun låses i 2 min. Før der kan komme en timeout på opgaven.

Databaser håndterer deadlocks ved at fjerne muligheden for no preemption. Det sker når en transaktion har siddet fast i et givent stykke tid i en database, så får den en timeout, som man også kalder for en deadlock timeout. Når dette sker, så går databasen ind og analyserer på de transaktioner, som sidder fast. Derefter analyserer databasen på, hvilken transaktion der har færrest locks, da det typisk er dem, som har lavet mindst arbejde og derefter aborterer denne. Normalt når man afbryder en transaktion, så burde det løse et deadlock problem, men det kan forekomme, at flere transaktioner skal afbrydes før dette lykkes.

## Delkonklusion

I dette distribuerede system vil der anvendes optimistisk samtidighed, det betyder, at man implementerer et lavt niveau af isolation level og anvender et "check" i disse metoder, som har risiko for race conditions. Dette check giver muligheden for at undgå race conditions og hvis "checket" fejler, aborteres metoden, trækker eventuelle ændringer tilbage og prøver igen. Derfor vil der være stor opmærksom på styrken af transaktionerne(ACID) og hvor højt et isolation level der menes at være nødvendigt at implementere.

# Sikkerhed

I dette kapitel vil der blive beskrevet, hvilke former for angreb der kunne være til fare for systemet og løsninger på disse angreb og hvordan at passwords vil blive håndteret.

## Sensitiv data

Systemets sensitive data består af brugeres private information username, password og e-mailadresse og for en Premium brugers navn, username, password og email. Derudover opbevares der information om film, serier og skuespillere i databasen. Systemets Premium brugere skal betale for ekstra funktionalitet i systemet, men der opbevares ikke dankort oplysninger fordi de kun skal bruges under transaktionen og slettes derefter fra systemet. En anden mulighed for betaling er en tredjepart service som Paypal, så der heller aldrig indtastes kortoplysninger i klienten.

## Attack vectors

I dette afsnit vil der blive beskrevet, hvilke angreb der kan være til fare for dette program og hvordan de bliver håndteret.

## SQL injection

SQL injection er SQL kodning i et tekstfelt, som brugeren selv har adgang til for eksempel, hvis brugeren taster et username ind og derefter den skadelige SQL kode 'drop table X'.

Konsekvensen af et sådan angreb kan være at miste dele eller hele databasen, hvilket ville være det værste, der kunne ske for dette system fordi der bliver opbevaret så store mængder data.

For at undgå denne type angreb benyttes der prepared statements det vil sige, at SQL kode ikke kan gå direkte fra UI -> Kontrol laget -> database.

## Broken authentication and session management

Broken authentication er udnyttelse af dårlig kodet godkendelse af username og password og session management er URL specifikke sessions eksempel "sessionID=1312312".

Konsekvensen af dette angreb er, at brugere kan få adgang til andre brugeres konti, hvilket ikke er alvorligt i dette system.

For at forsvare imod dette angreb sørges der for en god godkendelse for eksempel til password gendannelse og session kontrol. Det skal forstås sådan, at hvis en bruger klikker gendan password på en hjemmeside, så skal password helst sendes til en e-mail. Dårlig gendannelse defineres som for eksempel et spørgsmål "hvad hedder din hjemby" og hvis hjemmesiden så bare viser passwordet, hvis der er rigtigt svar, så har man en broken authentication.

## XSS

Cross-site scripting er når data eller en data request for lov til at komme ind i en 'trusted' web applikation gennem en 'un-trusted' kommunikation for eksempel en web request.

Konsekvensen af dette angreb er, at angriberen kan få adgang til brugerens konto eller omdirigere brugerens trafik. Da der er kommentare i dette system, så ville en bruger kunne taste et script i kommentarfeltet.

Løsningen på dette er, at teksten på server siden bliver renset, så alle HTML tags bliver taget væk.

## Insecure direct object references

Insecure direct object references sker når der er et system, som giver direkte adgang til et specifikt objekt alt efter et brugerinput.

Konsekvensen af dette angreb er sårbar data.

Løsning på denne type angreb er at hvis en anmodning indeholder et id'er på objekter, som en bruger ikke skal have adgang til det, så skal server siden skal tjekke om de objekter er tilgængelige for brugeren før serveren tillader, at brugeren udfører denne handling.

## Security misconfiguration

Security misconfiguration er simple efterladte huller i systemet efter sikkerhedstest for eksempel en admin-admin bruger, der har været brugt til test, men som ikke er fjernet i den udgave af programmet som bliver udgivet.

Konsekvensen af dette er, at angribere får adgang til systemet eller dele af det gennem disse efterladte huller.

For at undgå denne svaghed er det vigtigt, at efter test i systemet ryddes der op i programmets anomalier.

## Sensitive data exposure

Sensitive data exposure er ikke et specifikt angreb, men en fælles betegnelse for adgang til sensitivt data gennem forskellige metoder. For eksempel Bruteforce, Man-in-the-middle eller dekryptering.

Konsekvenserne af dette kan varierer alt efter om, det er en specifik bruger, en admin eller bare databasen som angribes.

Forsvaret er at overveje potentielle sikkerhedshuller og lukke dem før udgivelse.

## Storing passwords

I dette afsnit vil der blive beskrevet hvordan dette system håndtere password i forhold til hashing og saltning.

## Hashing

En kryptografisk hash funktion er en speciel gruppe af hash funktioner, der har visse egenskaber, som gør den egnet til brug i kryptografi. Det er en matematisk algoritme, der kortlægger data af vilkårlig størrelse til en smule streng med en fast størrelse og er designet til at være en envejsfunktion, det vil sige en funktion som ikke er mulig at vende om. Den eneste måde at genskabe input data fra en ideel kryptografisk hashfunktion på er at forsøge en brute-force søgning af mulige input, for at se om de matcher eller bruge et Rainbow tabel med matchede hash, løsningen på dette kunne være saltning.

Den ideelle kryptografiske hash funktion har fem hovedegenskaber:

- Det er deterministisk, så det altid ender ud med samme hash resultat af en given tekst.
- Det er hurtigt at beregne hash værdien af en given tekst.
- Det er umuligt at generere noget mening fra sin hash-værdi undtagen ved at prøve alle mulige kombinationer af tekst.
- En lille ændring i en tekst skal ændre hash værdien så omfattende, at den nye hashværdi fremstår uden sammenhæng med den gamle hashværdi.
- Det er umuligt at finde to forskellige tekster med samme hashværdi.



## Saltning

I kryptografi er et salt en tilfældig data, der bruges som et ekstra input til en envejsfunktion, der "hashes" typisk et kodeord eller en adgangskode. Saltet bruges til at beskytte adgangskoder i databasen. Historisk blev en adgangskode lagret i ren tekst på et system, men i løbet af tiden blev der udviklet yderligere sikkerhedsforanstaltninger for at beskytte brugerens adgangskode mod at blive læst af eventuelle personer, som måtte bruge disse koder til ondt brug.

Et nyt salt genereres tilfældigt for hver adgangskode. I en typisk indstilling er saltet og adgangskoden sammenkædet og behandlet med en kryptografisk hash funktion og det resulterende output, dog ikke den oprindelige adgangskode, lagres med saltet i en database. Hashing giver mulighed for senere autentificering uden at holde og dermed risikere plaintext adgangskoden, hvis authentication databasen er kompromitteret.

Saltet forsvarer imod ordbogsangreb eller mod deres hashed ækvivalent, et forudberegnet rainbow table angreb. Da saltet ikke skal huskes af mennesker, kan de gøre størrelsen af rainbow table angreb, der kræves til et vellykket angreb, umuligt stort set uden at lægge en byrde på brugerne. Da saltet er forskellige i hvert tilfælde beskytter de også almindeligt brugte adgangskoder eller de brugere, der bruger samme adgangskode på flere websteder, ved at gøre alle saltede hash forekomster til samme kodeord forskelligt fra hinanden.

## Delkonklusion

Opbevaring af kodeord sker på sigt med saltning og hashing og andre nævnte sikkerhedsrisiko bliver der også taget højde for. I systemet er der ikke mange sensitive data for eksempel sammenlignet med en bank. Konsekvensen af de fleste angreb er ikke alvorlige, da systemet bliver bygget på en sådan måde, at der ikke vil blive opbevaret data, som kunden ikke ønsker. En bruger kan for eksempel være anonym i systemet, hvis brugerens email ikke er kædet sammen med personlig information. I systemet bliver der på sigt også implementeret muligheden for premium brugere, som får adgang til ekstra funktionalitet. Denne bruger kan ikke være anonym, da brugeren skal betale for premium funktionaliteterne.

De værste angreb systemet kunne komme ud for er angreb, som beskadiger eller sletter film og serie databasen. Bruger tabellerne er ikke kritiske, fordi det er nemt og hurtigt at oprette en bruger, men der kommer til at være enorme mængder data i film og serie tabellerne, som kan være svært at genskabe. Her bliver en hyppig back-up relevant.

# Test

For at teste systemet er der blevet brugt programmerings mønsteret AAA (arrange, act, assert). Som kan ses på figur 10.



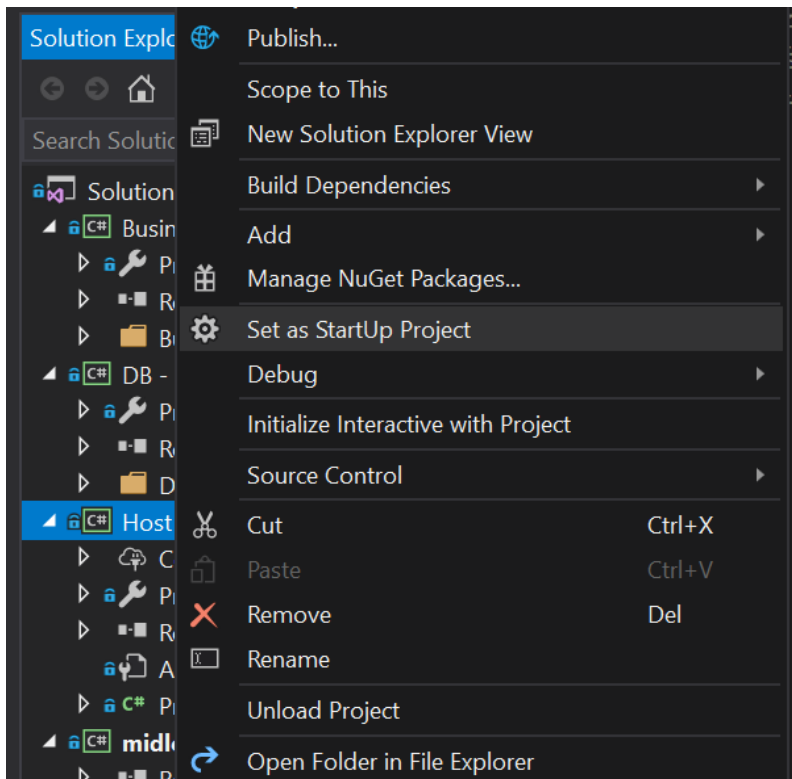
```
65 [TestMethod]
66 | 0 references | Andreas, 3 hours ago | 1 author, 1 change
67 public void LoginConfirmation()
68 {
69     Console.WriteLine("TestAddToMyList initiated");
70     Console.WriteLine("-----");
71
72     //Arrange
73     Service1 ss = new Service1();
74     User user = new User
75     {
76         Password = "password",
77         UserName = "Linaqx"
78     };
79     User u = new User();
80
81     //Act
82     u = ss.LoginConfirmation(user);
83
84     //Assert
85     Assert.AreEqual(1, u.Id);
86
87     Console.WriteLine("TestAddToMyList complete");
88     Console.WriteLine("-----");
89     Console.ReadLine();
90 }
```

Metoden, som kan ses fra billedet, tester om en bruger eksistere i databasen og kan derfor logge ind i web klienten. Under 'Arrange' arrangeres der de data, som skal benyttes under 'Act' i denne metode oprettes der en ny bruger, der stemmer overens med en allerede eksisterende bruger i systemet. Herefter under 'Act' handler vi med den information, som er sat op i 'Arrange'. Til sidst under 'Asser' sammenlignes det resultat der fås retur fra systemet med det forventede resultat, hvis testen er udført korrekt, går testen igennem. Der er ikke dette projekt ikke lavet test for alle funktionaliter. Dette er planlagt til senere versioner.

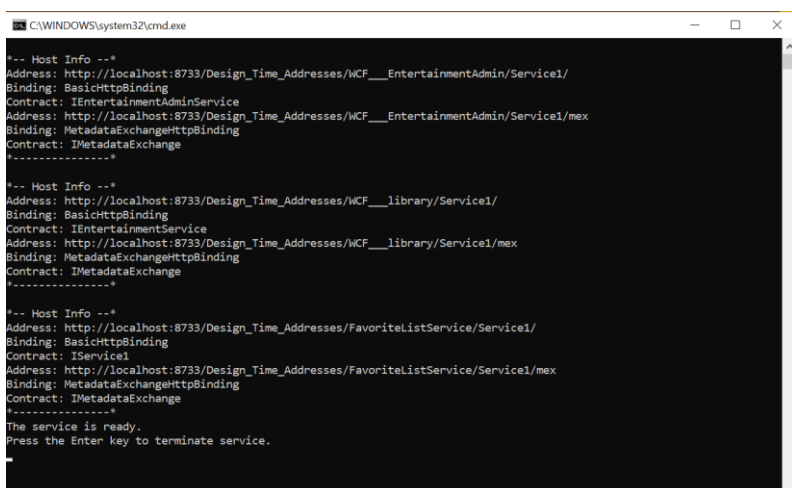
# Program guide

Hvis du vil sikre dig et program med frisk data, så kørs Flixnet Setup på Kraka Database med Server Navn: kraka.ucn.dk, Login: dmab0917\_1026423 og Password: Password1!

Step 1: Åben Visual Studio i administratortilstand. Åben herefter solutionen “Projekt 3 - WCF” hvis ikke allerede projektet “Host” er højreklik på dette og vælg “Set as startup Project” Dette kan ses på figur 11.



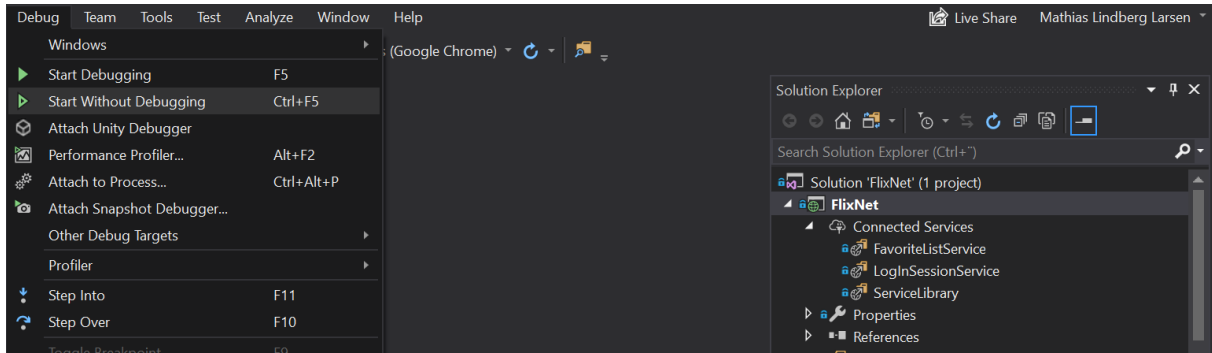
Step 2: Kør herefter programmet uden debugger (Ctrl+F5) kommandoprompt (CMD) burde nu åbne. Kan ses på figur 12.



Nu skal endnu en instans af visual studio åbnes, denne gang er administrator tilstand ikke nødvendig for at køre web klienten åben solutionen “FlixNet” for at se program guide for native klienten gå til Native client.

## Web klient

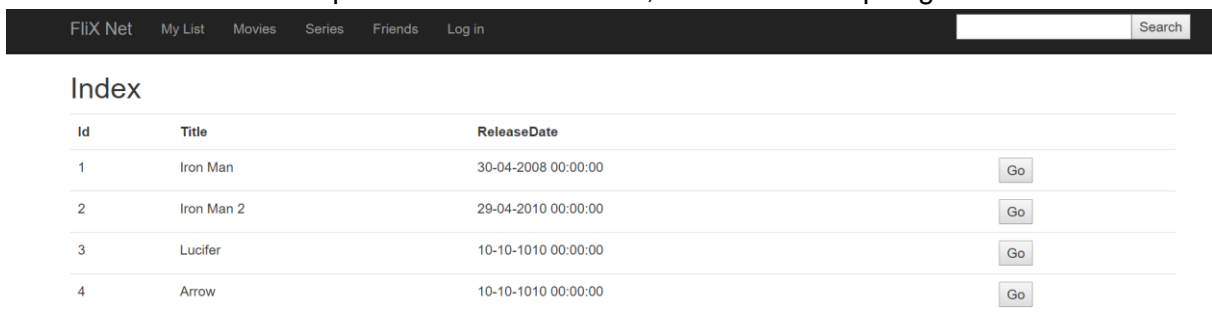
Step 3a: efter at step 1 og 2 er fuldført skal du nu køre web klienten uden debugger (Ctrl + F5) Dette kan ses på figur 13.



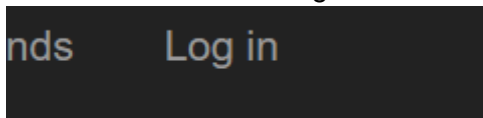
Step 4a: Nu har du følgende valgmuligheder

1. Log in
2. gå til film fra index
3. se dine favorit lister
4. skrive en kommentar på en film

Den første side du bliver præsenteret for er index, hvilket kan ses på figur 14.



Herfra skal du klikke Log in for at bruge resten af systemet, som kan ses på figur 15.



Log in siden ser sådan ud, som ses på figur 16

## LogIn

Herfra kan du frit vælge mellem de tre brugere som allerede eksisterer, se bruger data. Der anbefales, at du benytter brugeren Linaqx da der er mest eksisterende information, som kan ses på figur 17.

## LogIn

Klik nu på “Log in” du vil blive redirected tilbage til index, der vil nu prøves mulighed “2” gå til film fra index, vælg enten “Iron man” eller “Iron Man 2” de resterende valgmuligheder kan ikke vælges da de er serier, som kan ses på figur 18.

### Index

Id	Title	ReleaseDate	
1	Iron Man	30-04-2008 00:00:00	<input type="button" value="Go"/>
2	Iron Man 2	29-04-2010 00:00:00	<input type="button" value="Go"/>

På næste billede er Iron man 2 valgt og hentet fra databasen, som kan ses på figur 19.

### Movie

**Genre** Action  
**Title** Iron Man 2  
**Country** USA  
**Language** English  
**ReleaseDate** 29-04-2010 00:00:00  
**StoryLine** With the world now aware of his identity as Iron Man, Tony Stark must contend with both his declining health and a vengeful mad man with ties to his fathers legacy.  
**FilmingLocation** New York  
**Information** Some more awesome information!!

**Comments**  
 Add a comment

Du kan nu udføre valgmulighed 4. Skrive en kommentar på en film. Skriv din ønskede kommentar i tekstfeltet og klik submit, hvilket kan ses på figur 20.

Add a comment

Skriv her

Submit

Kommentaren kan nu ses under film tabellen, som kan ses å figur 21.

## Comments

super god film

Skriv her

Der kan nu prøves mulighed 3. Se mine favorit lister, klik her på "My List" i den øverste bjælke, som kan ses på figur 22.

FliX Net

My List

Movies

Series

# Index

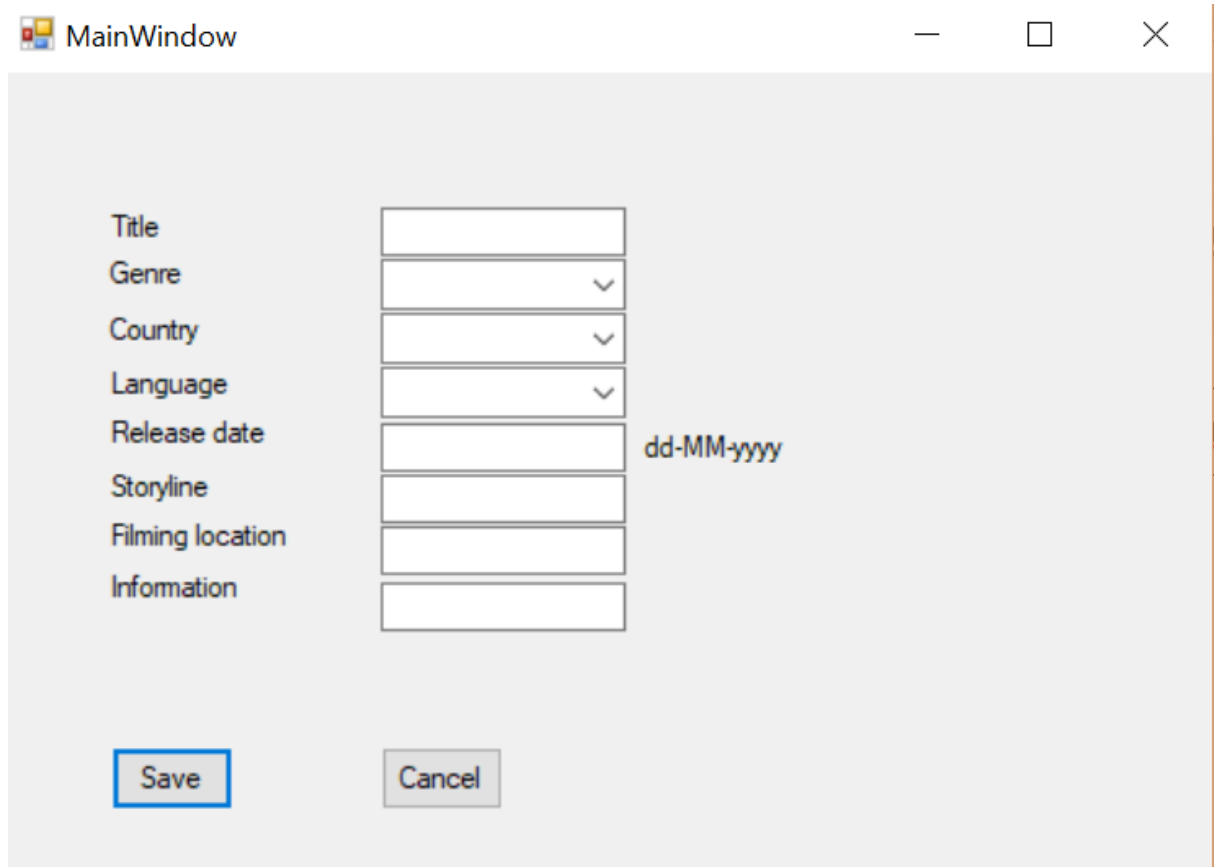
Dette er den nuværende funktionalitet i web klienten. Den kan kort fortalt hente fra databasen, sammenligne log in oplysninger og rette i databasen (comments). sikkerheden i systemet er stadig lavt der er planlagt at lave forbedringer af dette i senere versioner. password i for eksempel ikke hashed og salted og der er ikke lavet et forsvar for cross site scripting.

Bruger data som kan ses på tabel 5.

Brugernavn	Password	aktiv
Linaqx	password	ja
Ytte	1234	ja
KatrineVM	taellertilfirepaajapansk	ja

## Native klient

Fra step 2. åben nu en ny visual studio og åben solution “Projekt 3” og kørs projektet uden debugger. Vinduet der åbnes kan ses på figur 23.



The screenshot shows a Windows application window titled "MainWindow". Inside the window is a form with the following fields and controls:

- Title**: A text input field.
- Genre**: A dropdown menu with a downward arrow.
- Country**: A dropdown menu with a downward arrow.
- Language**: A dropdown menu with a downward arrow.
- Release date**: A text input field with a date format hint "dd-MM-yyyy" to its right.
- Storyline**: A text input field.
- Filming location**: A text input field.
- Information**: A text input field.
- Buttons**: "Save" and "Cancel" buttons at the bottom left.

Native klient kan indsætte en ny film til databasen, genre, language og country hentes fra databasen således de ikke bliver forkert udfyldt af de administrerende brugere.

## Konklusion

Projektets primære formål var først og fremmest at besvare problemformuleringen, som havde til formål at afdække det materiale, som gruppen havde fået undervisning i, i løbet af semestret. Det er lykkedes gruppen at lave et distribueret system, som håndterer problematikker med hensyn til sikkerhed. Samtidig overvejelser er afdækket teoretisk i rapporten og afdækket begrænset i programmet. Undervejs har gruppen anvendt teorier og metoder fra undervisningen til at løse de udfordringer, som de har mødt undervejs i udviklingsfasen.

# Litteraturliste

## Bøger:

Larman, Craig, Applying UML and Patterns, an introduction to Object-oriented analysis and design and the Unified Process, third edition

Kniberg, Henrik og Skarin, Mattias, Kanban and Scrum, making the most of both

Sommerville, Ian, Software Engineering, 10. edition, Pearson, 2015

Kniberg, Henrik, Scrum and XP from the Trenches, 2. edition 2015

Cohn, Mike, User Stories Applied, For Agile Software Development Addison-Wesley

Miles, Rob, C# Programming: <http://www.csharpcourse.com/> DK-students

Stallings, William, Operating systems: Internals and design principles, 8th edition

## Hjemmesider:

<http://www.tutorialsteacher.com/mvc/htmlhelper-textbox-textboxfor> (Sidst set den 12/12 2018)

<https://www.lifewire.com/definition-of-protocol-network-817949> (Sidst set den 12/12 2018)

<https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-2017> (Sidst set den 13/12 2018)

<https://www.asp.net/mvc> (sidst set den 17/12 2018)

[:https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-2017](https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-2017) (sidst set den 17/12 2018)

<https://www.lifewire.com/transmission-control-protocol-and-internet-protocol-816255> (Sidst set den 15/12 2018)

<https://www.lifewire.com/transmission-control-protocol-and-internet-protocol-816255> (Sidst set den 15/12 2018)

<https://www.lifewire.com/definition-of-protocol-network-817949> (Sidst set den 15/12 2018)

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions> (sidst set den 15/12 2018)



# Bilag

Bilag 1 - Domænemodel og relationelle diagram

Bilag 2 – Script til oprettelse af database