**Project Information**

Introduction:

Have you ever logged onto Netflix and had no idea what to watch next? There’s just so many choices! Choosing what to watch next can be difficult -- but it doesn’t have to be! We’re here to give you the next generation of the online streaming experience with Nextflix! Today, people have so many more important decisions to make, but this shouldn’t be one of them! Nextflix offers a personalized decision making algorithm that will help you find your next favorite show or movie! Nextflix not only keeps track of what the user has watched, but also what actors the user is a fan of. Nextflix is then able to find new content for the user to enjoy. Nextflix allows users to maximize their subscription of Netflix by making sure they always have something to watch.

Nextflix does this by linking TV shows and movies based on the lead actors/actresses, genre, release date, director, and other similar shows in a database. This database allows the show-finding algorithm to access the next ideal show for the user. With a click of a button the user is off to the next show.

\*\*Nextflix is not responsible for loss of productivity or increased occurrences of “lazy afternoons”. All lost sleep is at the discretion of users.

Requirements:

Users must not be able to add/remove actor, movie, tv show, similar\_to

Users must not be able to remove other users

Users must be able to edit their own user data

Users must be able to add new movie, or tv show to has\_watched tables

Users must be able to like movies/tv shows in has watched tables

Users must be able to find similar movies/tv shows

Users must be able to create user account

Users must have passwords

Application must make sure that SQL attacks are prevented

Users must be able to add to fan\_of

Users must not be able to edit other users fan\_of or has\_watched entries

Users must be able to export their recommended show/movies

Users must be granted admin status from another admin

Users must be at least 13 years of age

Users must agree to Nextflix’s terms of use

Passwords must be hashed for security purposes

Users must enter password in order to change preferences

All user data must be removed when their account is deleted

The user must be able to change their password

The application must provide the user with a random movie

**The Design Process**

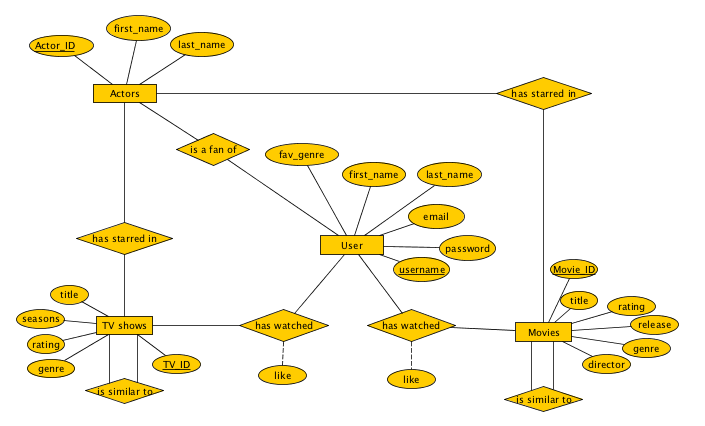
For the overall design of the site, we wanted to keep it very clean and simple. The whole point of the site is to make the process of finding a new show easy, so we utilized pictures of the movies and scroll bars in order to minimize reading. The user instantly has twenty some options of suggested shows when logged in. These shows are based off of the favorite genre and actors of the user, so the probability of a hit is high. The scroll bars are horizontal because the user naturally reads the moves from left to right. This allows for a more user friendly experience and speeds up navigation through the site. If the user really want to know more about a movie title, they can click on the picture of the movie, and then they are brought to a screen with all of the important info on a movie or show. We decided to move to a new page because it is less likely for the user to be distracted. It gave us plenty of room to largely display all of the information to increase readability.

Additionally, security was of the utmost priority to us. There may not be any sensitive personal information in our database, but we value our users’ peace of mind. No one wants to change their password when their account has been hacked, so we took every measure we could to prevent this. In the login process, prepared statements are used in order to prevent SQL injection attacks. The passwords are also hashed with a MD5 algorithm from the timed they are first entered. This is done so that even if an attacker can access database, they still won’t know the passwords of our users. As well, the user is required to enter their password whenever they want to change their preferences. This was done to protect the user’s settings. It also prevents inadvertent changes.

We chose to create a web application because Netflix is a web application, and our users will be able to easily access Nextflix when they are done with their previous show or movie. We chose PHP because our database is already in PHPmyAdmin, and interfacing with it was really easy. We looked at possibly using Django, but ultimately decided otherwise.

We used special functions like stored procedures in order to select which movies should be recommended. When on the home screen, the what to watch section uses SQL to select out movies in the user’s favorite genre that they have not seen yet. The other special function is another stored procedure used to select a random show. The motivation for this feature is for when the user wants to explore new parts of Netflix, but doesn’t know what to search. The random button selects a random movie or show from our database, exposing the user to new, fresh content.

E-R Diagram



Database Schema

CREATE TABLE TVShows (

Title VARCHAR(255),

Seasons INT,

Genre VARCHAR(255),

Rating DOUBLE,

TVID INT primary key);

CREATE TABLE Movies (

title varchar(50),

movie\_id int primary key,

rating int, genre varchar(50),

director varchar(30),

release\_date date);

CREATE TABLE Users (

username VARCHAR(255) primary key NOT NULL ,

first\_name VARCHAR(150) NOT NULL,

last\_name VARCHAR(250) NOT NULL,

email VARCHAR(300) NOT NULL,

password VARCHAR(500) NOT NULL,

fav\_genre varCHAR(150) NOT NULL

);

CREATE TABLE Actors (

Actor\_ID char(5) primary key not null,

first\_name char(30),

last\_name char(30) not null);

CREATE TALBE starin\_movie

(Actor\_ID char(5),

movie\_id int);

CREATE TABLE starin\_tv (

TVID INT not null,

Actor\_ID char(5) not null );

CREATE TABLE Fan\_Of (

Username VARCHAR(255) not null ,

Actor\_ID char(5) not null);

CREATE TABLE similar\_movies

(movie\_id int,

similar\_to\_movie\_id int);

CREATE TABLE Watched\_show (

Username VARCHAR(255) not null,

TVID int not null );

CREATE TABLE Watched\_movie (

Username VARCHAR(255) not null,

movie\_id int not null );

Proof of TNF

User:

A: username

B: password

C: email

D: first\_name

E: last\_name

F: fav\_genre

A->BCDEF

Actors:

G: Actor\_ID

H: first\_name

I: last\_name

G->HI

Movies:

J: Movie\_ID

K: title

L: rating

M: release

N: genre

O: director

J->KLMO

TV shows

P: TV\_ID

Q: title

R: seasons

S: rating

T: genre

P->QRST

Is fan of

A: username

G: Actor\_ID

A->G

Has starred in movie

G: Actor\_ID

J: Movie\_ID

G->J

Has starred in TV

G: Actor\_ID

P: TV\_ID

G->P

Has watched movie

A: username

J: Movie\_ID

U: liked\_movie

A->JU

Has watched TV

A: username

P: TV\_ID

V: liked\_TV

A->PV

Is similar to movie

J: Movie\_ID

W: Movie\_ID2

J->W

Is similar to TV

P: TV\_ID

X: TV\_ID2

P->X

Canonical Cover(Fc)

A->BCDEFGJUPV

G->HIJP

J->KLMOW

P->QRSTX

There are no redundancies or extra relationships to take out since we design the database to only have relevant attributes and relationships. The four main entity sets can be used as candidate keys to access most other relevant data. For example, the username (A) has not only all of their personal information tied to them, but also all of the movies/TV shows they have watched. While having these four big tables is nice because they have all of the relevant information we could need, we are leaving the database as ten tables because most of the information is not necessary for common queries, and it makes the data base easier to manipulate. 3NF asserts that the data is relevant to the table that it is in, and having four huge tables would violate this. Each of the 10 tables is in 3NF because they all have one candidate key and only contain relevant information. Some of them, like is similar to TV could be turned into multivalue tables, but we did not want to have null values because that would waste space and ultimately not be very scalable.

**Evaluation of Product**

...How did we test our application/database

Sample Data and Queries

... SQL code here