Status Report 1

Group 1

Tristan Allen, Daniel Carter, Will Cox, Josiah Jackson

3/24/24

1. Introduction

1.1 Highlights

- Our work items were the following:
    - Create a basic database schema to hold data for movies, user information, and ratings within a MySQL database.
        - Movie table would hold a unique movie id, movie title, genres, cast, directors, movie description, movie keywords, runtime and release date.
        - User table would hold a unique user id, username and password.
        - Ratings table would hold a user id, movie id, user rating, and the date the movie was rated.
    - Create a data importer script that imports the data of interest for each movie from a CSV dataset to populate our MySQL database “movie” table.
    - Implement basic two piece User Authentication api.
        - A “sign-up” api to add username and password to the database with password encryption for added security
        - A “login” api to search for username in database and check given password against hashed password

    - Design User Interface for each page and create wireframes to visualize.
        - Additionally show wireframes to customers and get feedback
    - Create a basic functional ML algorithm that can take a movie as input and then output similar movies based on director, cast, genre, and movie keywords.
- The following work items were completed
    - Database Schema
    - Data Importer
    - User Authentication
    - User Interface Wireframes
- The following work items are still being completed/ have major defects
    - Machine Learning Algorithm

1.2 Changes

We made one major change to our system from the project proposal into the first status report.

3/10/2024 - Decided to implement user authentication in backend instead of frontend

Initially we were planning on using AuthO with React to login users. After some thought we decided to implement user authentication utilizing the flask_login module in our backend to more easily manage user sessions.

2. Customer-Iteration Goals

Product Backlog:

- Search functionality that allows user to filter movies by Genre, Director, Cast, Runtime, Rating.
- User is able to rate movies that they have previously watched
- User is able to like a specific movie they have previously watched
- Login functionality/User Authentication
- Implement a database to store movies and ratings
    - Data schema to structure a MySQL database
    - Data importer to populate the database
- Develop a machine learning algorithm that is able to give personalized recommendations based on a user profile
- Establish a connection between User Interface and Flask Backend
- Create a User Interface with four pages
    - Sign Up/Login
    - Home
    - Recommendations
    - Profile

Customer's Desired Overall Experience:

- Our customer wants to be able to keep a record of past movies and their enjoyment of them, while also being able to quickly find new movies that are aligned with their interests.

Sprint Backlog:

- Implementing database
    - Data importer

    - Database schema
- Login functionality/User Authentication
- User Interface
    - Wireframes for each page
    - Prototype frontend
- Machine Learning Algorithm

We chose these items for our spring backlog because we felt that these items set a foundation for our system. These are all critical parts of our system that will likely evolve as our development continues. For example, our ML algorithm will be continually tuned throughout the entire development process so we want to establish a baseline as early as possible in the development process.

2.1 Use Cases

Name: Rating a Movie

Goal: A user gives a movie they have previously watched a rating

Actors: User, Flick Finder

Basic Flow:

1. User opens Flick Finder

   {network connected}

2. User logs into their account

   {User Authenticated}

3. User searches for movie that they have previously watched

   {database queried}

4. System returns movies that match Users search criteria

5. User clicks on movie they want to rate
6. User rates movie

   {database updated with new rating}
7. User is notified their rating was successful

   {return to home}

Specific Alternate Flow:

At **{Database queried}** if there are no movies that match the users search criteria, the system displays “No movies found” and the basic flow is resumed at **{return to home}**.

Bounded Alternate Flow:

At any point between **{network connected}** and **{database updated with new rating}** if the network connection is disrupted, system displays “connection lost” and the basic flow is resumed at **{return to home}**.

Name: Receiving a list of recommendations

Goal: A user receives a concise list of movie recommendations

Actors: User, FlickFinder

Basic Flow:

1. User opens FlickerFinder

   {network connected}
2. User logs into their account

   {User Authenticated}
3. User selects the recommend option

   {Machine Learning Algorithm is executed}

4. FlickFinder returns a concise list of movie recommendations for the user
5. The User can browse movie options and select the movie they would like to watch

   {Return to Home}

Specific Alternate Flow: At **{Machine Learning Algorithm is executed},** if the machine learning algorithm fails and returns an error, the system displays "Sorry, we can not provide movie recommendations at this time" and basic flow is resumed at **{Return to Home}.**

Bounded Alternate Flow: At any point between **{network connected}** and **{return to home}** if the network connection is lost, the system displays "connection lost" and basic flow is resumed at **{Return to Home}.**

Name: Browse through movie options

Goal: A user browses through a list of movie options based on specific topics

Actors: User, FlickFinder

Basic Flow:

1. User opens FlickFinder

   {Network Connected}

2. User logs into their account

   {User Authenticated}

3. User accesses homepage

   {Homepage loads on screen}

4. User presses browse button

   {Browsing page displays movies}

5. User selects specific topic to browse

   {Database queried}

6. User clicks on movie to see information about it

   {Information about movie displayed}

7. User logs out

   {Return to login screen}

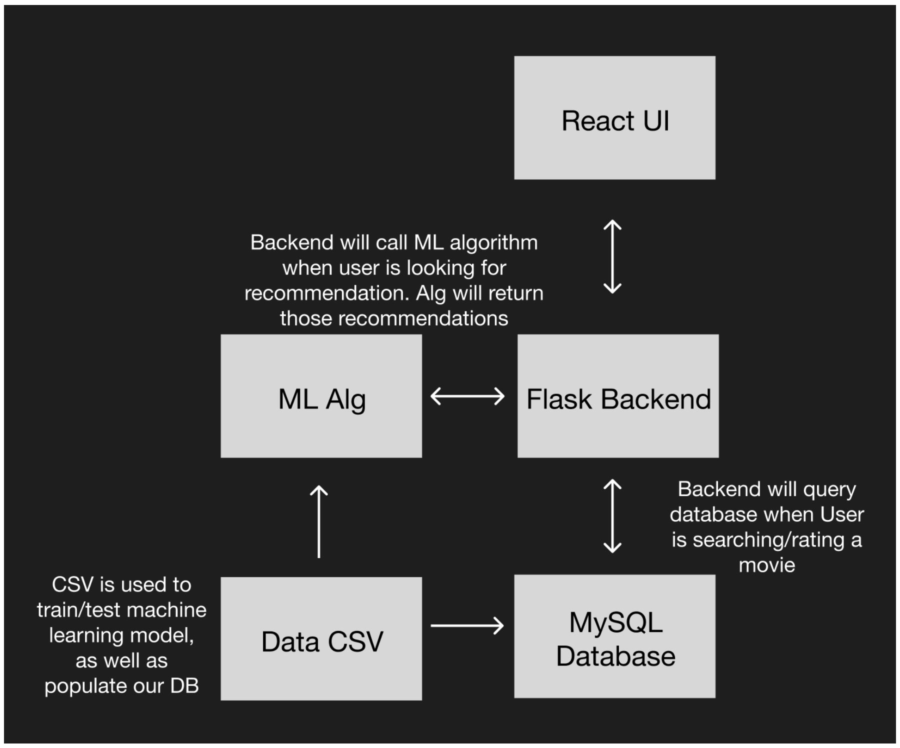
Specific Alternate Flow:

At **{Information about movie displayed}**, if there is no information about the movie, the system displays “Sorry, no information found”, and basic flow resumes at **{Browsing page displays movies}**.

Bounded Alternate Flow:

At any point between **{Network Connected}** and **{Information about movie displayed}** if network connection is lost, the system displays “Connection Lost” and basic flow is resumed at **{Return to login screen}**.

3. System Description


React UI
Backend will call ML algorithm when user is looking for recommendation. Alg will return those recommendations
ML Alg
Flask Backend
Backend will query database when User is searching/rating a movie
CSV is used to train/test machine learning model, as well as populate our DB
Data CSV
MySQL Database

Our current system is composed of 4 distinct parts: A ReactJS frontend, a flask backend, a SVD machine learning algorithm, and a MySQL database. We go into further detail about these parts of our system in the next section

4. Current Status

We currently have four main parts to our system. We have basic functionality for our frontend, backend APIs, and database, but we are still trying to implement our movie recommender algorithm.

We implemented a MySQL database that includes three tables: movies, users, and ratings. Currently for each movie our movie table holds a unique movie id, movie title, genres, cast, directors, movie description, movie keywords, runtime and release date. We implemented a data importer script using python that reads through our CSV datasets and populates our movie table. There are a couple movies that are missing values for their runtime and/or release date. We currently substitute a “-1” for a missing runtime and “0001-01-01” for a missing release date. The users table currently holds a unique user id, username and password. Lastly the reviews table holds a user id, movie id, user rating, and rate date.

Our current frontend leverages ReactJS and is a basic webpage mostly used to test our backend APIs. For this iteration we implemented 5 pages: a home screen, a sign-up screen, a login screen, a recommendation page, and a movie page. Our home screen simply includes the title of the website and the version. Currently the sign-up screen and login screen are visually very similar with a form that takes a username and password (submitting these forms call different APIs in the backend). Our recommendation page currently is hardcoded to show the same three movies (since we do not have our recommender algorithm functioning at the moment) but requires a user to be logged in to access the page (if a user is not logged in no movies will be recommended). From the recommender page if the user selects a movie, the movie page is displayed and includes the movie's title, description and genres. After talking with customers and getting feedback, we created wireframes for our UI to help focus what we want to implement moving forward.

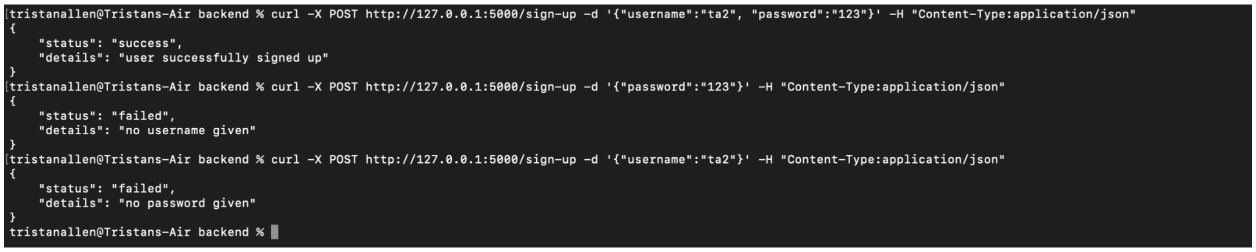
Our backend utilizes multiple flask modules to implement a RESTful API that currently handles 5 main functionalities: sign up a user, login a user, logout a user, find user movie recommendations (currently hardcoded), and get data for a certain movie. Sign-up takes a POST request with username and password to populate the user table in our database (for added security the password is encrypted before being put into the database). Login takes a POST request with username and password to check the database for an existing username and compares the given password against the encrypted password. Logout takes a GET request and ends the current user session. Recommendations takes a GET request that queries the database for 3 movies (with our algorithm currently not completely operational the chosen movies are hardcoded). Lastly a movie takes a GET request and queries the database for the given movie and returns the movie title, description, and genres. Our backend also manages user sessions and prevents users not logged in from accessing some pages (currently just the recommendations page).

Our ML algorithm is a singular value decomposition (SVD) model. This is the standard model for recommender systems. Our model is trained with 23 features, which includes things like director, genre, cast, etc.It is a hybrid recommender model, meaning that it utilizes aspects of content based recommending and collaborative filtering. By using a hybrid model we are able to recommend movies on their similarity to other movies, as well as how other users rate movies.

4.1 Screenshots

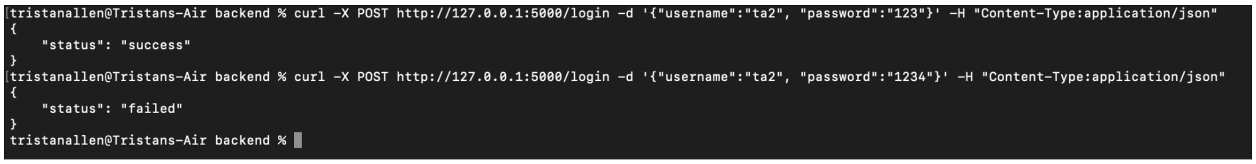
API screenshots:

- Signup API: the below screenshot shows the POST command for signing up a user and the return statuses associated. The first is correct usage, the second no username is given and the third no password is given.



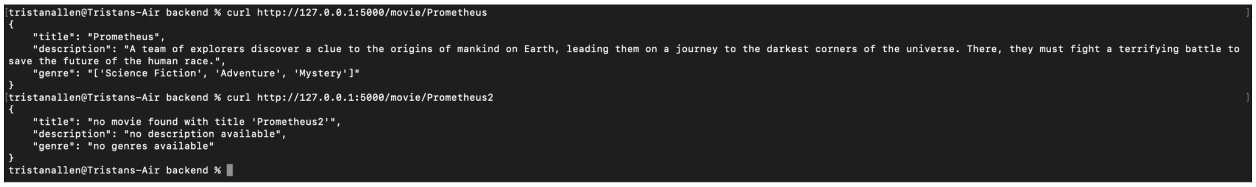
```
[tristanallen@Tristans-Air backend % curl -X POST http://127.0.0.1:5000/sign-up -d '{"username":"ta2", "password":"123"}' -H "Content-Type:application/json"
{
    "status": "success",
    "details": "user successfully signed up"
}
[tristanallen@Tristans-Air backend % curl -X POST http://127.0.0.1:5000/sign-up -d '{"password":"123"}' -H "Content-Type:application/json"
{
    "status": "failed",
    "details": "no username given"
}
[tristanallen@Tristans-Air backend % curl -X POST http://127.0.0.1:5000/sign-up -d '{"username":"ta2"}' -H "Content-Type:application/json"
{
    "status": "failed",
    "details": "no password given"
}
tristanallen@Tristans-Air backend %
```

- Login API: the below screenshot shows the POST command for logging a user in and the associated return statuses. The first was correct credentials and the second was an incorrect password.



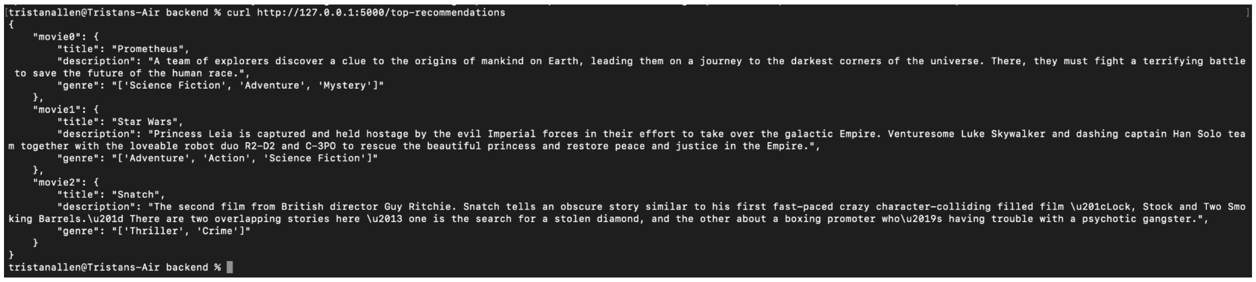
```
[tristanallen@Tristans-Air backend % curl -X POST http://127.0.0.1:5000/login -d '{"username":"ta2", "password":"123"}' -H "Content-Type:application/json"
{
    "status": "success"
}
[tristanallen@Tristans-Air backend % curl -X POST http://127.0.0.1:5000/login -d '{"username":"ta2", "password":"1234"}' -H "Content-Type:application/json"
{
    "status": "failed"
}
tristanallen@Tristans-Air backend %
```

- Movie API: the below screenshot shows the GET command for getting information for a movie. The first is for the movie Prometheus and the second is a movie that does not exist, Prometheus2. The information is pulled from the "movie" table in our database.



```
tristanallen@Tristans-Air backend % curl http://127.0.0.1:5000/movie/Prometheus
{
    "title": "Prometheus",
    "description": "A team of explorers discover a clue to the origins of mankind on Earth, leading them on a journey to the darkest corners of the universe. There, they must fight a terrifying battle to
save the future of the human race.",
    "genre": "['Science Fiction', 'Adventure', 'Mystery']"
}
tristanallen@Tristans-Air backend % curl http://127.0.0.1:5000/movie/Prometheus2
{
    "title": "no movie found with title 'Prometheus2'",
    "description": "no description available",
    "genre": "no genres available"
}
tristanallen@Tristans-Air backend %
```

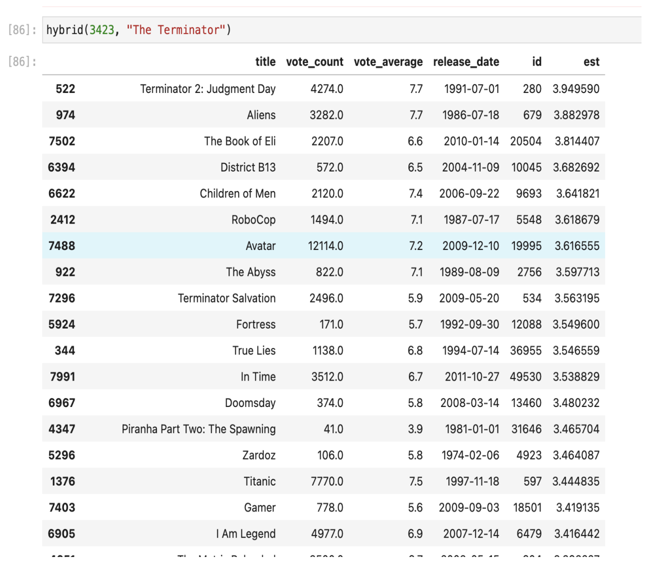
- Recommendation API: the below screenshot shows the GET command for getting the list of recommended movies. Currently it is hardcoded to show the same three movies until we implement our ML algorithm. This is the return value when a user is logged in.



```
tristanallen@Tristans-Air backend % curl http://127.0.0.1:5000/top-recommendations
{
    "movie0": {
        "title": "Prometheus",
        "description": "A team of explorers discover a clue to the origins of mankind on Earth, leading them on a journey to the darkest corners of the universe. There, they must fight a terrifying battle
 to save the future of the human race.",
        "genre": "['Science Fiction', 'Adventure', 'Mystery']"
    },
    "movie1": {
        "title": "Star Wars",
        "description": "Princess Leia is captured and held hostage by the evil Imperial forces in their effort to take over the galactic Empire. Venturesome Luke Skywalker and dashing captain Han Solo tea
m together with the loveable robot duo R2-D2 and C-3PO to rescue the beautiful princess and restore peace and justice in the Empire.",
        "genre": "['Adventure', 'Action', 'Science Fiction']"
    },
    "movie2": {
        "title": "Snatch",
        "description": "The second film from British director Guy Ritchie. Snatch tells an obscure story similar to his first fast-paced crazy character-colliding filled film \u201cLock, Stock and Two Smo
king Barrels.\u201d There are two overlapping stories here \u2013 one is the search for a stolen diamond, and the other about a boxing promoter who\u2019s having trouble with a psychotic gangster.",
        "genre": "['Thriller', 'Crime']"
    }
}
tristanallen@Tristans-Air backend %
```

ML screenshot:

- The screenshot is showing the output of our recommender function. The way it currently functions is that a user and a movie are inputted into the function and recommendations are given based on the movie and user profile.



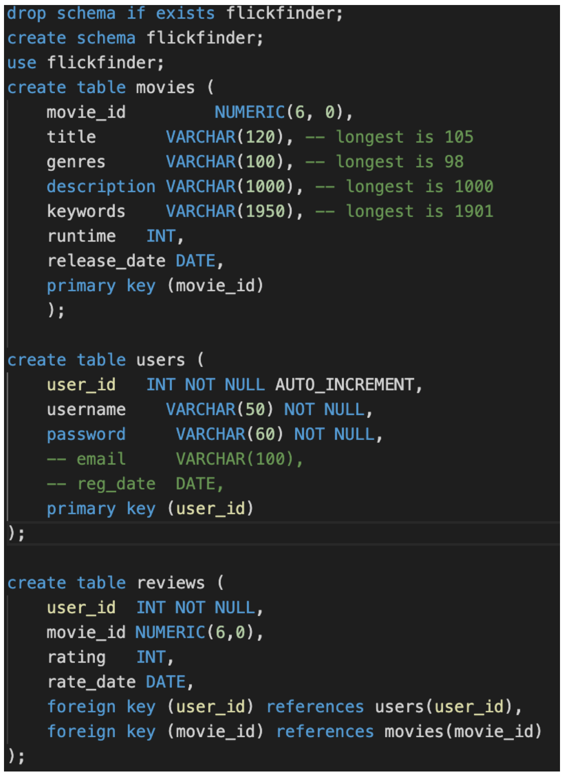
```
[86]: hybrid(3423, "The Terminator")
```

[86]:

	title	vote_count	vote_average	release_date	id	est
522	Terminator 2: Judgment Day	4274.0	7.7	1991-07-01	280	3.949590
974	Aliens	3282.0	7.7	1986-07-18	679	3.882978
7502	The Book of Eli	2207.0	6.6	2010-01-14	20504	3.814407
6394	District B13	572.0	6.5	2004-11-09	10045	3.682692
6622	Children of Men	2120.0	7.4	2006-09-22	9693	3.641821
2412	RoboCop	1494.0	7.1	1987-07-17	5548	3.618679
7488	Avatar	12114.0	7.2	2009-12-10	19995	3.616555
922	The Abyss	822.0	7.1	1989-08-09	2756	3.597713
7296	Terminator Salvation	2496.0	5.9	2009-05-20	534	3.563195
5924	Fortress	171.0	5.7	1992-09-30	12088	3.549600
344	True Lies	1138.0	6.8	1994-07-14	36955	3.546559
7991	In Time	3512.0	6.7	2011-10-27	49530	3.538829
6967	Doomsday	374.0	5.8	2008-03-14	13460	3.480232
4347	Piranha Part Two: The Spawning	41.0	3.9	1981-01-01	31646	3.465704
5296	Zardoz	106.0	5.8	1974-02-06	4923	3.464087
1376	Titanic	7770.0	7.5	1997-11-18	597	3.444835
7403	Gamer	778.0	5.6	2009-09-03	18501	3.419135
6905	I Am Legend	4977.0	6.9	2007-12-14	6479	3.416442

Database screenshot:

- Schema: Our current database schema for the movies, users, and reviews tables. We have unique IDs for users and movies to differentiate them, and both IDs are foreign keys in the reviews table.



```
drop schema if exists flickfinder;
create schema flickfinder;
use flickfinder;
create table movies (
    movie_id         NUMERIC(6, 0),
    title       VARCHAR(120), -- longest is 105
    genres      VARCHAR(100), -- longest is 98
    description VARCHAR(1000), -- longest is 1000
    keywords    VARCHAR(1950), -- longest is 1901
    runtime   INT,
    release_date DATE,
    primary key (movie_id)
    );

create table users (
    user_id   INT NOT NULL AUTO_INCREMENT,
    username    VARCHAR(50) NOT NULL,
    password     VARCHAR(60) NOT NULL,
    -- email     VARCHAR(100),
    -- reg_date  DATE,
    primary key (user_id)
);

create table reviews (
    user_id  INT NOT NULL,
    movie_id NUMERIC(6,0),
    rating   INT,
    rate_date DATE,
    foreign key (user_id) references users(user_id),
    foreign key (movie_id) references movies(movie_id)
);
```

4.2. Tests

Database tests:

- SQL schema file can be run with no errors and database is created with tables “movies”, “users”, and “ratings” and their respective columns.
    - # mysql FlickFinder < "flick_finder_schema.sql"
- Data importer file can be run with no errors and the “movie” table is populated with movie data.

  - # python3 data_importer.py

Frontend tests:

- Home page renders
- Signup and login pages both render with a form that takes a username and password as input.
- Logout button ends the current user session.
- Recommendation page renders three movie links to their respective movie pages. If a user is not logged in then no movies are listed.

API tests:

- Sign-up POST request takes a username and password as input and populates the “user” table with the username given, the encrypted password, and a unique user id. Returns a status success if username and password are both given, failure otherwise.
- Login POST request takes a username and password. Returns status success if the username and encrypted password are in the database, failure otherwise.
- Top-recommendations GET request return 3 movies, currently hardcoded as “Prometheus”, “Star Wars”, and “Snatch”. If a user session is active, the three movies are returned, otherwise “no movie available” is returned.
- Movie GET request takes a movie title and returns the movie title, description, and genres associated with the movie.

User Authentication tests:

- User’s password should be encrypted before being put in the database.
- User session should stay active until the logout button is pressed.

Acceptance Tests:

- Given a user has liked/disliked a movie when they navigate to their profile page then the movie is in its respective section (liked/disliked)
    - Input: User likes a movie on Flick Finder
    - Output: Movie is placed into the liked category on the users profile
    - Pass/Fail: If the movie is not placed into the liked category the test is failed
- Given a user has watched a movie when the user clicks on that movie then they are able to rate that movie.
    - Input: A movie that has not been rated by the user
    - Output: Movie has been rated by user
    - Pass/Fail: If the user is unable to rate a movie then the test is failed.
- Given a user is interested in a movie, when the user clicks on that movie, then they are able to see information about that movie (genre, director, cast, etc.).
    - Input: A movie
    - Output: When the movie is clicked on the user is able to see more information about the movie
    - Pass/Fail: If there is no information that pops up when a movie is clicked on the test is failed.
- Given a user watches a movie and really enjoys it, when the user selects it as one of their favorites, then it is displayed as one of their favorites on their profile.
    - Input: A movie that a user has designated as one of their favorite movies
    - Output: It is displayed in the favorites category on the user profile

    - Pass/Fail: If the movie is not tagged and moved into the favorite category on the users profile the test is failed.
- Given a user has inputted movies that they liked when they hit the recommend button then they will receive a short list of movies that are similar to their liked movies.
    - Input: User
    - Output: List of movies that are recommended to the user
    - Pass/Fail: If the user is not recommended movies then the test is failed.

5. **Project Management**

Date	Description
3/10	Decided to implement user authentication in backend instead of frontend

6. **Review and Retrospective**
    - Our group was productive during this sprint and set a great foundation for our system. All of our work items are either fully functional or implemented and have minor defects that will be fixed in our next sprint.
    - We ran into problems with our dev environments and not being able to import the correct packages
    - In regards to our ML algorithm, we had issues transferring the script from a Jupyter notebook to a python script. Many of the imports were unable to resolve when transferred to a python file, which caused the script to exit with errors when ran.

- We need to do more research to understand why this is happening. We believe that there is an issue with the python interpreter at least on one of our local machines. We have been looking into creating a virtual environment to run these scripts in so that we can have a consistent environment across all of our machines.
- In the next iteration we plan to make sure everyone is working in the same environment at the beginning of the sprint so we do not spend a large amount of time trying to troubleshoot errors that are caused from not having the right python interpreter, packages, or libraries installed.

7. Team Management

- Team Roles:
  - Tristan Allen: Scrum Master/Developer
  - Will Cox: Product Owner/Developer
  - Daniel Carter: Developer
  - Josiah Jackson: Developer
- Team Member Contributions:
  - Tristan Allen: User Authentication/ Data Importer/ Database Schema
  - Daniel Carter: Database Schema/ Data Importer
  - Will Cox: ML Algorithm
  - Josiah Jackson: UI Wireframes
- Communication lacked throughout our first sprint. Due to differing schedules, along with spring break, we found it difficult to find times to meet and discuss things that hindered progress on work items, which left individuals on their own

to figure it out. We also were unable to find a consistent meeting time during the week.

- We plan to set aside an hour each Wednesday to meet for a stand up meeting to discuss what is going well/ not going well in person. By meeting in person we feel that everyone will be more inclined to communicate openly about the current spring.
- We would suggest communicating not only when issues arise in the development process, but also when tasks have been completed. By communicating when your work items have been completed you are letting the rest of your team know that you have more capacity to help them with their tasks.

8. **Goals for the Next Iteration**

Product Backlog:

- Search functionality that allows user to filter movies by Genre, Director, Cast, Runtime, Rating.
- User is able to rate movies that they have previously watched
- User is able to like a specific movie they have previously watched
- Develop a machine learning algorithm that is able to give personalized recommendations based on a user profile
- Establish a connection between User Interface and Flask Backend
- Create a User Interface with four pages
    - Sign Up/Login
    - Home
    - Recommendations

    - Profile

Sprint Backlog:

- Search functionality that allows user to filter movies by Genre, Director, Cast, Runtime, Rating.
- User is able to rate movies that they have previously watched
- User is able to like a specific movie they have previously watched
- Connecting ML algorithm to frontend UI
- Tuning the ML algorithm

A potential challenge that we face for this iteration is the quick turnaround between status report 1 and status report 2. This period is much quicker so as a group we will all need to set aside more time to get our work items done for this iteration. To overcome this our group needs to be honest about our capacities for work, so we do not plan to do more work than we are actually capable of doing.