Video link: https://youtu.be/XqGZN_YAEYg

1) An overview of the function of the code (i.e., what it does and what it can be used for).

The code is used as a chrome extension that recommends 10 movies to the user based on similarity. To be able to use this, a user would need to input IMDb movie details page links of the movies they like/ have watched and the code will then respond by returning back 10 movies (title and plot) back to the user.

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

First we will start with the front end:

The extension uses web technologies consisting of React JS with Typescript, HTML, and CSS. Adding @types/chrome using yarn, the app creates the boilerplate to configure itself for a chrome extension. Listing the app into chrome extensions is easy and requires building the app, then "load unpacked" the build/ folder under chrome://extensions with developer mode switched on.

The frontend will get the url of the current tab using chrome's builtin methods and place it onto a controlled form displayed to the user, where the user then can press the "Bookmark Movie Url" to list the current URL into the set of bookmarked movie urls stored in local storage of the user's browser. Immediately upon listing the URL, the frontend fetches the api with params of chaining the urls together with "data" for each of their keys. The json response received is then stored in an array of objects called "recommendations", which is then used to create "MovieCard" components for each of the 10 movies. These "MovieCard" components are then displayed on the chrome extension pop up with their title and plot.

Then, the backend:

The code uses a flask server in order to receive requests. Upon starting up the server, it will first fill the idf dictionary which is pre-computed before using the movies_genre.csv dataset. It will also read in the data of the movies itself such as title, plot, genre, etc… Then, it will get the tf-idf vectors for each movie. Afterwards, the server will be up and running. Upon receiving a post request with a list of movie links as a parameter, the code will first obtain the plot of each movie via web scraping on their IMDb link. After it is done scraping, it will then get the tf-idf vectors for each movie. Afterwards, it finds the similarity between movies that were given and movies in the dataset and recommends the top 10 most similar movies in the dataset and returns it back to the front end.

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

First, we need to build the app. Go to the cs410-final-project/ directory and run

yarn run build

This creates the build/ folder in the same directory. Go to chrome://extensions with developer mode on and click the "Load unpacked" and select this build/ folder. Afterwards, pin the chrome extension on the top.

To run backend, do python flask_api.py or py flask_api.py or py3 flask_api.py in the backend directory.

NOTE: Additionally, when bookmarking a new movie, you will need to close the extension and open again for the bookmarks to be updated with the new movie added. This is due to the asynchronous nature of the chrome extension that we couldn't find a work-around.

4) Brief description of contribution of each team member in case of a multi-person team.

Putra handled the backend side of things such as setting up the flask server, precomputing of idf, algorithm to get the tf_idf of movies and algorithm of scraping. Ardel handled the frontend side such as setting up the chrome extension, creating the pages, components, and elements for the extension, and marrying together the front end with the backend api. The design and implementation of the recommender function was done by both group members.