Mini Project 2 Design Document

A. Overview and User Guide

This project provides basic functions for searches and updates to MongoDB document stores about movies and movie people. Making sure the files "name.basics.tsv", "title.basics.tsv", "title.principals.tsv", "title.ratings.tsv" are in the same directory, run “tsv-2-json.py”. This will convert all the tsv files into JSON files. Then, run “load-json.py”, inputting the port MongoDB server is listening to. This program will load all the JSON data from the 4 files into a database called 291db (it will drop all previous data if a database called 291db already exists). Now you can run the main program “project.py”. The program starts in the menu screen where the user has 6 options: search for a title, search for genres, search for cast/crew members, add a movie, add a cast/crew member, or exit the program. Searching for title prompts users to enter the keywords, separated by spaces. The program returns all movies that have those keywords in the title. The user can then view more information about a result, which will show its rating, vote count, cast members and their corresponding characters. Then it will return to the menu. Searching for genre will prompt the user to enter a genre and the minimum vote count. The program will then display all the movies and their ratings that are in that genre and have the minimum vote count. Then it will return to the menu. Searching for a cast/crew will prompt the user to enter the name of the person. Then the program will return all the people with that name, their professions, and the movies they played a character or had a job in. Then it will return to the menu. Adding a movie will prompt the user to enter a unique ID, title, start year, running time, and list of genres, and add it to the database. Then return to the menu. Adding a cast/crew member will prompt the user to enter a crew/cast member’s ID, a movie’s ID, and a category, and then add it to the database. Then it will return to the menu. The user can also exit the program from the menu.

B. Design Components

Tsv2json: opens a .tsv file and gets the column headers. Then goes line by line and parses the information into a dictionary with the column header as the keys, and the data corresponding to that header as values. All the dictionaries are stored in a list, which is then turned into a JSON file that is saved in the same directory.

LoadJson: connects to the MongoClient and opens a database called 291db. Then it opens each of the four collections, drops its contents, and then adds new contents from the JSON files. Then it creates an index for some of the document fields.

Search_movies: The user provides space separated keywords. If the word is a number, it makes sure that either the year or a word in the title. If it’s a word, then it just must exist in the title. Once searched, you can input one of the corresponding numbers and a rating, vote count, and a list of actors with their characters appear.

Search_genres: the program asks the user for a genre, and the minimum vote count (making sure the vote count is numeric). Then it performs an aggregation of title_principals and title_basics to get a corresponding vote count and rating and matches the genre to the movie. Votes must be greater than the minimum count.

Search_cast: the program prompts the user to enter the name of a cast/crew member. It queries the name_basics collections to find a list of cast/crew members with the same name. Then for each result, it gets the name, professions, and movies they are a part of. For each movie they are a part of, the title_principals collection is queried to find all documents that have the same title ID as the movie, and same name ID as the result, where either they have at least one character played, or a job in the movie. All the information is then displayed.

Add_movie: prompts the user to enter the ID, title, year, start time, and genres of the movie. The program then checks to see if the entered year and start time are numeric values, and if the ID is unique (does not exist already in the database). If all the information is valid, it inserts a document with all the information into the title_basics collection.

Add_cast: prompts the user to enter a crew ID, and title ID. It then checks to see if both the crew ID and title ID exist, throwing up an error if they do not. The ordering is set to the largest ordering listed for the title plus one by querying title_prinicipals to find all documents with matching title ID and sorting in descending order of ordering. The first document will have the highest ordering. If no documents are found from the query, the ordering is set to 1. The data is then added to a new document in the title_principals collection.

C. Testing Strategy

Our general strategy consisted of first doing a simple test by looking at the database and manually figuring out either what the function/query should return or what the database should look like after the function/query is run for some given (valid) values. We then ran the program with the given values and checked if the program results matched our manual results. We then did more complex tests, such as trying invalid values (IDs that do not exist, entering letters when numbers are expected and vice versa, entering invalid characters), again doing a manual calculation before and comparing with results.Then we tested all the related functions together. Testing the menu: first we tried entering the values that correspond to each function and checked if it redirected to the (valid) corresponding menus. We then tried entering invalid values such as numbers and options that did not exist, and made sure an error message was thrown. Testing search_title: first we queried the database to find all the movies that should be returned for a given set of keywords. Then we input the same keywords in our program and checked to see if it was returning the same results. Then we tried entering keywords that we knew would not find any results, and made sure that our program was not finding any results. Testing search_genres: first we queried the database to find all the movies that should be returned for a given genre and vote count. Then we input the same keywords in our program and checked to see if it was returning the same results, and that the ordering was in descending

order of rating. Testing search_casr: first we queried the database to find all the cast members that should be returned for a given name. Then we input the same name in our program and checked to see if it was returning the same results, and that all the proper movies and characters/jobs for each movie were being displayed. Testing add_movie: first we input a unique title ID and proper values into our program. Then we checked the database with MongoDB Compass to see if the document was properly added. Then we tried to input invalid values, such as non-unique title ID, and non-numeric runtimes and start years. Testing add_cast: first we input valid values, such as existing title IDs and crew IDs. Then we checked the database with MongoDB Compass to see if the document was properly added. We made sure to test this for documents that had previous orderings and also documents with no titles listed in table_principles. All inputs for functions are also tested with basic error checking (such as invalid characters, and list numbers out of range). Most of the bugs we had were simple error checks, mainly entering list indexes out of range. These were solved by making sure the user could not enter those invalid indexes. Our biggest bug was that our queries for search movies and genres were taking too long. We fixed this issue by adding indexes for some of the fields when loading them onto the database.

D. Group Work Break Down

First we worked together to create a starting template which included the programs we were going to need, names of the primary functions, and some basic helper functions such as the menu. Faiyad created the tsv-2-json.py program (30 mins) while Mark wrote the load-json.py program (30 mins). Mark fully wrote the search movies function (1 hour). Mark wrote most of the search genres function as well (1 hour) while Faiyad wrote some error checking for it and did some minor debugging (20 mins). Faiyad wrote the majority of the search cast function (1 hour), while Mark did some debugging for it (30 mins). Faiyad fully wrote the add movies function (30 mins) and the add cast function (30 mins). Mark wrote some error checking for them (20 mins). We both spent 60 mins together to combine all our work and make sure everything was working and also ensure consistency between our work. We did 1 hour of testing and debugging together and 30 minutes on our own doing testing and another 30 mins doing miscellaneous debugging that resulted from our tests. For the report, Mark wrote the test strategies (25 mins) while Faiyad wrote the overview and user guide, and group work breakdown (40 mins). We each wrote the details of our own functions for the design component (30 minutes for each of us). We each worked on our own sections on different files but everytime we both finished a significant component we combined our work, using the Live Share extension on VS Code. To stay on track we both set realistic deadlines like “finishing X sections by Y day”, and kept each other accountable.