Justine:

Module: Sign up

Equivalence Classes:

Illegal inputs:

- Username < 4 characters, 15 characters < username
- Email: invalid email format
- Password < 8 characters , 80 characters < password

Legal inputs:

- 4 characters <= Username <= 15 characters
- Valid email format
- 8 characters <= password <= 80 characters

Username Test Cases:

- *Input*: username = cat
- *Output*: " field must be between 4 and 15 characters long"
- *Input*: username = DogsAreWayBetterThanCats
- *Output*: " field must be between 4 and 15 characters long"
- *Input*: username = four
- *Output*: input accepted

Email Test Cases:

- *Input*: email = user@gmail
- *Output*: "invalid email"
- *Input*: email = user.gmail.com
- *Output*: "invalid email"
- *Input*: email = user@gmail.com
- *Output*: input accepted

Password Test Cases:

- *Input*: password = 1234567
- *Output*: "field must be between 4 and 15 characters long"
- *Input*: password = abdjjvkddlvneuswdhvnsudovbsdbv8sdbvjskbu939bsdbv8b3800e8hvkjbdkbvsjkbvjsbdvbsvbud
- *Output*: "field must be between 4 and 15 characters long"
- *Input*: password = 12345678
- *Output*: input accepted

Module: Sign in

Illegal inputs:

- Username < 4 characters, 15 characters < username
- Password < 8 characters , 80 characters < password
- Username does not exist in our user table
- Password does not match expected password for entered username

Legal inputs:

- 4 characters <= Username <= 15 characters
- 8 characters <= password <= 80 characters
- Username exists in our user table
- Password matches password expected for entered username

Authentication Test Case:

- exists in our User table □Username = elonMusk , Password = elonMusk
- *Input*: username = elonM, Password = elonMusk
- *Output*: “invalid username or password”
- *Input*: username = elonMusk, Password = 12345678
- *Output*: “invalid username or password”
- *Input*: username = elonMusk, Password = elonMusk
- *Output*: user is directed to recommendation page

Username Test Cases:

- *Input*: username = cat
- *Output*: “ field must be between 4 and 15 characters long”
- *Input*: username = DogsAreWayBetterThanCats
- *Output*: “ field must be between 4 and 15 characters long”
- *Input*: username = four
- *Output*: input accepted

Password Test Cases:

- *Input*: password = 1234567
- *Output*: “field must be between 4 and 15 characters long”
- *Input*: password = abdjjvkddlvneuswdhvnsudovbsdbv8sdbvjskbu939bsdbv8b3800e8hvkjbdkbvsjkbvjsbdvbsvbud
- *Output*: “field must be between 4 and 15 characters long”
- *Input*: password = 12345678
- *Output*: input accepted

Babak:

- Tested the API using an external function called “api_read()” inside the movie_api.py to store the result to a text file called “api_data.txt” to observe the functionality of my call, reading the API with my specific calls over the query string to the API server. And then checking the stored data and comparing them with the expected data by using a tool inside the Visual Studio Code to query over my data and visually observe.
- Tested the query string sending from the server to the UI by using the flask route, and making sure the query string works properly. Adding a different query string

manually to the URL and check the expected result helped a lot in the debugging process.

- Tested the queries sent to the database using defined functions as boolean- and checking them against the expected result.
    - # insert_to_db()
    - # db.select('ariel')
    - # db.drop_M_TB()
    - # db.creat_db()

- Tested the JSON output using an external app called postman, check the API query string and all the necessary parameters.
- Tested the modified JSON output using a website called "https://jsonlint.com/" to verify the JSON object I was making after the modification, by feeding the website the result of my JSON file.
    - Legal output/input had to have a JSON format
- Validating the Valid movie object by checking the JSON objectmovie_data.get('status_code') != 34:

Joe:
Module: merging two databases into one database with 2 Tables using Flask-SQLAlchemy
Equivalence class: Indirect Output. The code that I wrote was equivalent to the examples in Flask-SQLAlchemy documentation
(https://flask-sqlalchemy.palletsprojects.com/en/2.x/quickstart/)

Initially, we were creating separate databases for our User and Movie classes. We saw this as inefficient and had to figure out how to merge the two databases. Luckily in my research I stumbled upon the above material and the code was easy to reproduce in our design and architecture. When I implemented the techniques from the documentation, Anna was able to use the tools provided by the VSCode Framework to look inside the database to insure the accuracy of the desired output.

Anna:

1. Functionality/module : Movie autocomplete
    - Equivalence Classes:
        - Legal inputs- all movies in our database up to 5
        - Illegal inputs - all movies not in our database or more than 5
        - Test case : legal -> finding nemo
        - Test case: legal -> f
        - Test case : illegal -> divergent

- Testing was quite simple for this module I simply typed in a movie that was legal and going to be recognized by our database, if it returned a dropdown of movies in our database then it was working properly. If a movie was illegal then we would not get a dropdown of any options. Movies would only get added if a dropdown option was available. This made sure users only selected movies within our database.

2. functionality/module: adding movies to favorites/watched lists
    a. Equivalence Classes:
        i. legal inputs - movies that user has selected as a favorite or watched/ movies within the relation database that maps users to movies
        ii. Illegal inputs - movies that were not selected as favorites or watched by user
        iii. Test case : legal -> any movie where button was clicked (heart/check)
        iv. Test case : illegal -> any movie where button was not clicked
    b. Testing for this module was just a matter of seeing if when the heart/checked button was clicked the movie would be added to the relevant carousel in the user home page. I did this by clicking a random movie on the recommended page and navigating to the user home page to see if the carousel was modified with the correct movie. I could have also checked the database to see if the movies and users were getting mapped together properly.
3. Functionality - recommend button (worked on with Justine + Joe)
    a. Equivalence classes :
        i. legal inputs: movies that user selected from dropdown, up to 5
        ii. Illegal inputs: nothing is illegal since we made it so that the user could only input legal movies by implementing the autocomplete dropdown
        iii. Test case ->(legal) finding nemo
        iv. Test case -> (legal) finding nemo, gladiator, blind chance, talk to her, dirty dancing
    b. Testing was a matter of clicking the button and seeing if the genre of the movie or movies given matched the genre of the recommended movies. So like all the movies recommended for the test case finding nemo should have either an animation genre or family genre.