**CS673 Software Engineering** 

**Team 6 - Blockbuster**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Joshua Shilts | Leader | *Joshua Shilts* | 9/11/2024 |
| Elizabeth Tyree | Design and Implementation Leader | *Elizabeth Tyree* | 9/11/2024 |
| Ricky Zheng | Configuration Leader | *Ricky Zheng* | 9/11/2024 |
| Alex Flinchum | QA Leader | *Alex Flinchum* | 9/11/2024 |
| James Zheng | Security Leader | *James Zheng* | 9/11/2024 |
| Elizabeth Tyree | Requirement Leader | *Elizabeth Tyree* | 9/11/2024 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **v1.0.0.** | [Joshua Shilts](mailto:jshilts@bu.edu) | **9/22** | **Creation** |
| **v1.0.1.** | [Elizabeth Tyree](mailto:elityree@bu.edu) | **10/6** | **Update** |
| **v1.0.2.** | [James Zheng](mailto:jamzheng@bu.edu)**/Alex Flinchum** | **10/7** | **Update** |
| **v1.0.3.** | [Elizabeth Tyree](mailto:elityree@bu.edu) | **10/7** | **Update** |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.30j0zll)

[Class Diagram](#_heading=h.1fob9te)

[UI Design (if applicable)](#_heading=h.3znysh7)

[Database Design (if applicable)](#_heading=h.2et92p0)

[Security Design](#_heading=h.tyjcwt)

[Business Logic and/or Key Algorithms](#_heading=h.3dy6vkm)

[Design Patterns](#_heading=h.1t3h5sf)

[Any Additional Topics you would like to include.](#_heading=h.4d34og8)

[References](#_heading=h.2s8eyo1)

[Glossary](#_heading=h.17dp8vu)

# Introduction

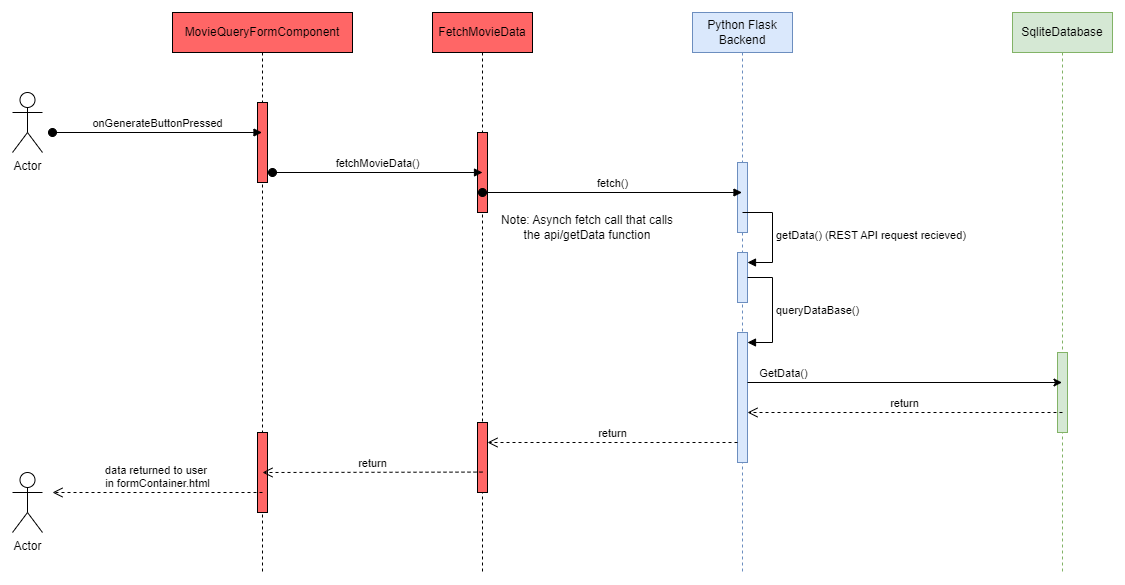
This document provided a brief overview of our software design and architecture.

# Software Architecture

Our software is broken down into three main components or microservices that is built within the Python Flask framework that utilizes Rest API calls to communicate between each service.

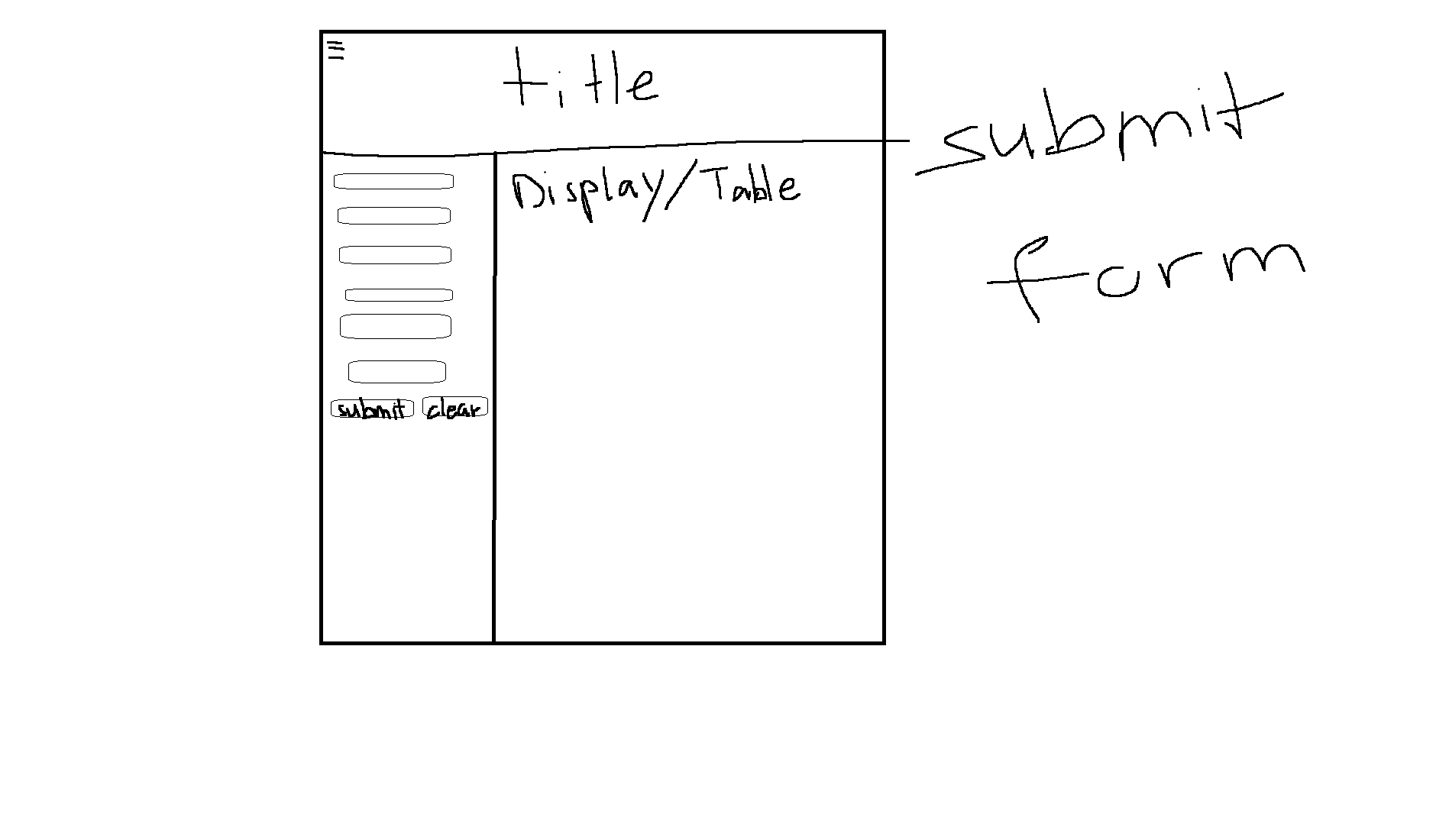
# Class Diagram

This is a sequence diagram that shows the flow of events, responses and data when a user submits a query by pressing the generate button. This diagram is expected to grow with more features if time allows to work on cosmetic features in the UI.



# UI Design

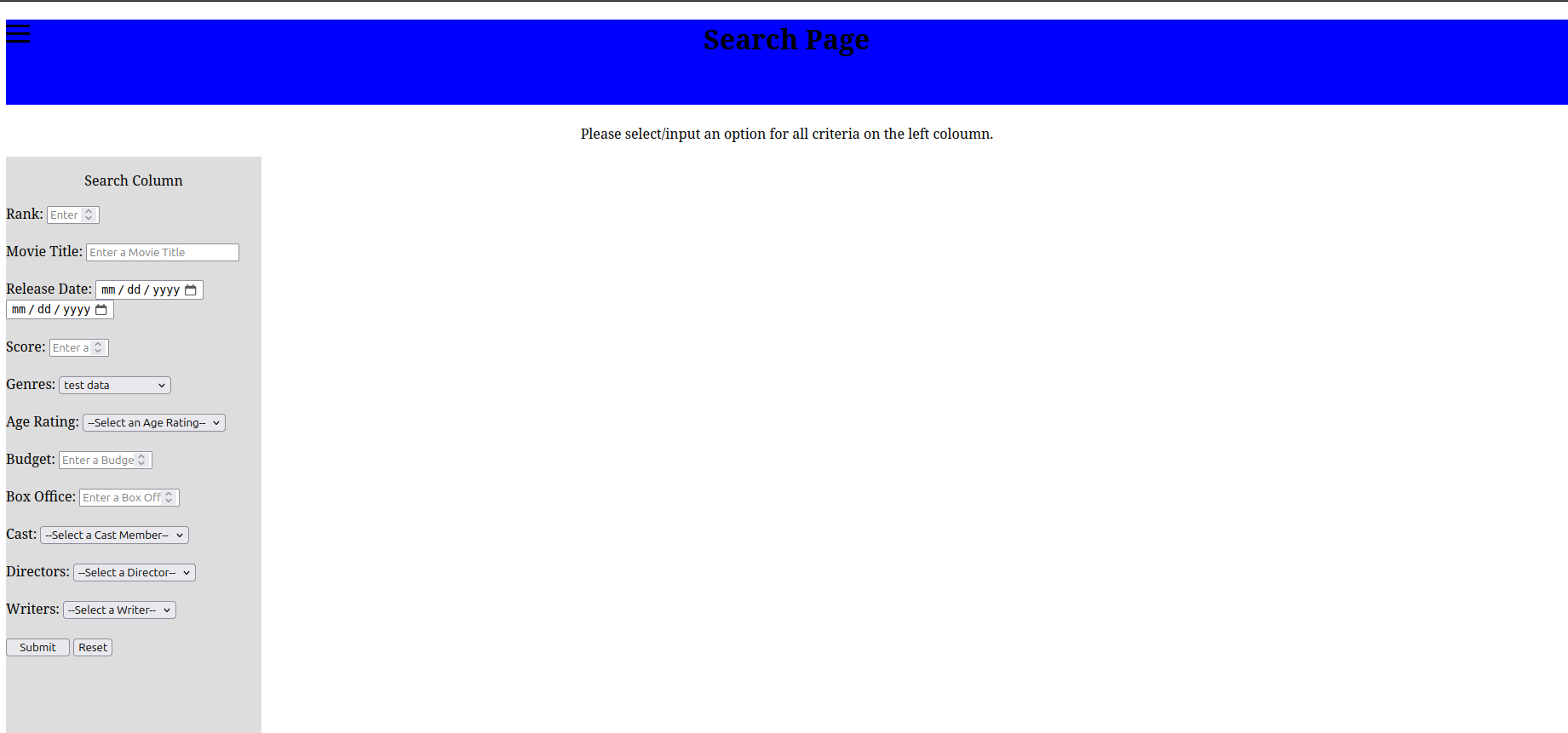
* + Previous UI mock up



UI design description (please see screenshot on next page):

* In the top right corner is a hamburger button function
  + That leads back to the home page
* The page is split into two different columns
  + The left column is the user input field that is used for search criteria
    - genres, age rating, cast, directors, and writers are dropdown fields
    - Movie title is a text input field
    - Score, rank, budget, and box office require an input type number
    - Release date start/end is a date input mm/dd/yyyy
    - Submit button sends all the criteria to the backend to retrieve the data the user inputted
    - Clear button that clears all the criteria fields
  + The right column is where we display the retrieved data based on the searched criteria
* A title for the page at the very top center
* Below the the title prompts the user to select/input to fill out all the criteria

Screenshot of UI (work in progress):



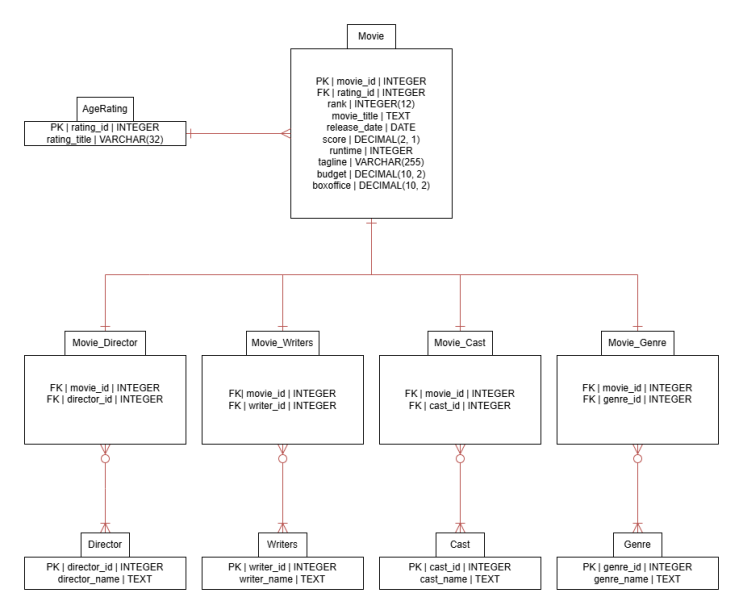
# Database Design

The Database design uses joint tables in order for movies to have more than one director, writer, cast, and genre associated with it

The age rating also gets its one table so that we do not have to repeat data within the movie table.

* There is one age rating for many movies
* There is one Movie\_Director entry for one movie
* There is one Movie\_Writers entry for one movie
* There is one Movie\_Cast entry for one movie
* There is one Movie\_Genre entry for one movie
* There is one or many Directors for zero or many Movie\_Directors entry
* There is one or many Writers for zero or many Movie\_Writers entry
* There is one or many Casts for zero or many Movie\_Casts entry
* There is one or many Genres for zero or many Movie\_Genres entry

SQLite Movies Database Design:



# Security Design

The security design at the moment is to utilize our pipeline and existing tools or plugins in github to find and address some vulnerabilities in our code. We plan to use PyLint and a static code analysis tool and pick several security defects in our code and create a bug and resolve that issue.

# Business Logic and/or Key Algorithms

N/A

# Design Patterns

We are attempting to use MVC in the frontend and also the observer pattern in React. That is everytime a state changes react will in hope will trigger an event that will call functions in the corresponding controller that will update the UI based on a user action such as a button press.

# References

# Glossary

**MVC -** Model View Controller

**REST API -** Representational State Transfer Application Interface

**UI -** User Interface