# Movie Review Database

CS 4347.502 | Database Systems

Professor: Jalal Omer

Team: Bennett Oppenheimer, Ahmad Bakhit Al Fayez, Fernando de Salvidea, Joseph Saber

## Introduction

This project is the result of our team's work to create a functioning and interactable database. Our idea was to create a database that contains the data of the top movies and all of their relevant information, such as actors, directors, descriptions, ratings, and more. Users can search for this information before they make their review. So, our project is a database of movies and user reviews. This report is an account of our work and will contain information relevant to our project. It will be separated into sections mirroring the table of contents.

System Requirements: Discusses the architectural overview, interface requirements, and functional/non-functional requirements of the database system.

Conceptual Design of the Database: Presents the Entity-Relationship (ER) model along with the data dictionary and business rules.

Logical Database Schema: Provides the schema of the database, SQL statements for schema construction, and expected database operations.

Functional Dependencies and Database Normalization: Identifies functional dependencies, normalization process, and normalized tables.

The Database System: Describes the installation process and user guide with step-by-step instructions along with screenshots.

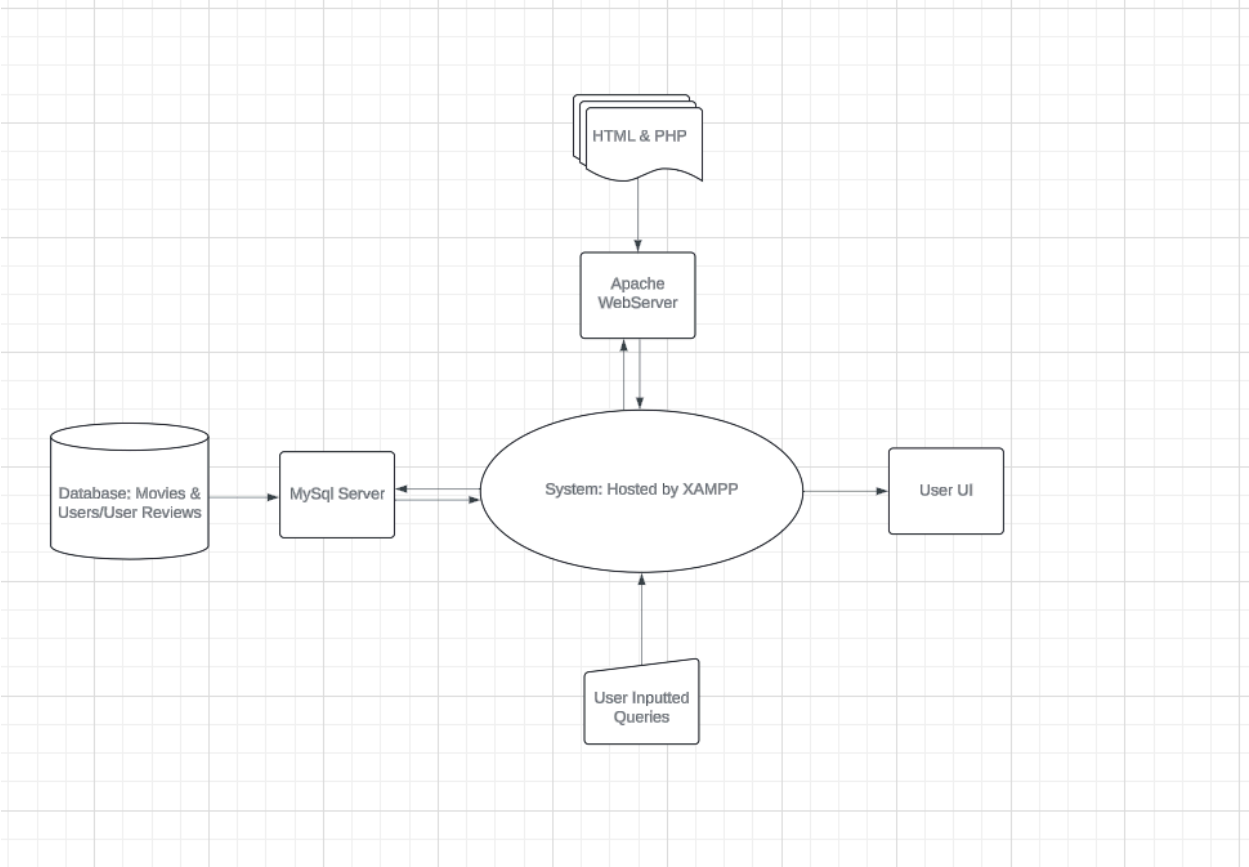
Suggestions on Database Tuning (optional): Offers suggestions for database tuning to enhance performance.

User Application Interface: Details the development of the system's user interface and user interaction.

Conclusions and Future Work: Summarizes the project outcomes, conclusions, and proposes future enhancements.

## System Requirements

Context Diagram:




System - The entire system is hosted using XAMPP. XAMPP is a free and open-source cross-platform web server solution stack package that uses multiple independent services to create a simple but effective experience. Our team utilized the Apache and MySQL services on this project.

Apache WebServer - The Apache web server was used to host our HTML files. The HTML was dependent on the PHP, so the PHP was stored alongside the HTML.

MySQL Server - The MySQL server was used to host our database and execute the queries.

User UI & User Inputted Queries - As mentioned the HTML is hosted by Apache, so our UI is accessible due to Apache. The queries are input through the UI, but sent to the MySQL server.

## Functional Requirements:

1. Movie Information Management:
    - The system should allow the addition, deletion, and modification of movie details such as title, release date, genre, director, actors, plot summary, etc.

2. User Review Management:
    - Users should be able to create, edit, and delete their reviews for movies.
    - Reviews should include ratings, comments, and timestamps.

3. Search and Filter Functionality:
    - Users should be able to search for movies based on various criteria such as title

4. User Authentication and Authorization:
    - The system should support user registration and login functionalities.
    - Administrators should have additional privileges for managing movies and user reviews.

5. User Interaction:
    - Users should be able to rate movies on a numerical scale and provide written reviews.

6. Data Integrity and Validation:
    - The system should enforce data integrity constraints to ensure accuracy and consistency of movie information and user reviews.
    - Input validation should be implemented to prevent invalid data entry.

## Non-Functional Requirements:

1. Performance:
    - The system should respond quickly to user queries and interactions, with minimal wait.
    - It should be capable of handling concurrent user requests efficiently.

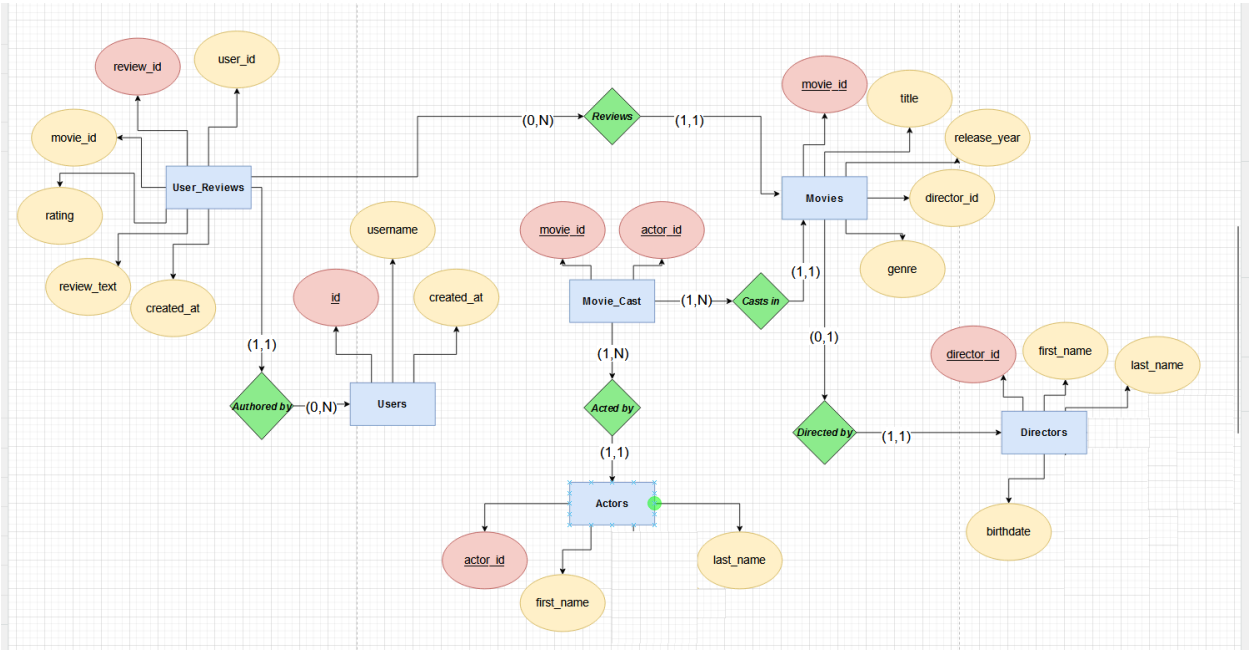
2. Scalability:
    - The system should be scalable to accommodate a growing number of movies, users, and reviews over time.

3. Reliability:
    - The system should be highly available and reliable, with minimal downtime.

4. Usability:
    - The user interface should be user-friendly

## Conceptual Design

ER Diagram:




Data Dictionary:

1. Movies Table:
    - movie_id (Primary Key): Unique identifier for each movie.
    - title: Title of the movie.
    - release_year: Year the movie was released.
    - director_id (Foreign Key): Identifier referencing the Directors Table, linking to the director of the movie.
    - genre: Genre of the movie.

2. Actors Table:
    - actor_id (Primary Key): Unique identifier for each actor.
    - first_name: First name of the actor.
    - last_name: Last name of the actor.
    - birthdate: Birthdate of the actor.

3. Directors Table:
    - director_id (Primary Key): Unique identifier for each director.
    - first_name: First name of the director.
    - last_name: Last name of the director.
    - birthdate: Birthdate of the director.

4. Movie_Cast Table:
    - movie_id (Foreign Key): Identifier referencing the Movies Table, linking to the movie in which an actor is cast.
    - actor_id (Foreign Key): Identifier referencing the Actors Table, linking to the actor cast in a movie.

5. User_Reviews Table:
    - review_id (Primary Key): Unique identifier for each review.
    - user_id (Foreign Key): Identifier referencing the Users Table, linking to the user who wrote the review.
    - movie_id (Foreign Key): Identifier referencing the Movies Table, linking to the movie being reviewed.
    - rating: Decimal value representing the user's rating for the movie.
    - review_text: Text containing the review written by the user.
    - created_at: Timestamp indicating when the review was created.

6. Users Table:
    - id (Primary Key): Unique identifier for each user.
    - username: User's username.
    - role: User's role.
    - created_at: Timestamp indicating when the user account was created.

## Business Rules:

1. Movies Constraints:
    - The title of a movie must be provided and unique.
    - Release year, director, and genre can be optional.
    - Actors and Directors Constraints:
    - Both first name and last name of actors and directors are required.

2. Movie Cast Constraints:
    - Both movie_id and actor_id must be provided, indicating the movie and actor cast.

3. User Reviews Constraints:
    - Each review must be associated with a user and a movie.
    - The user_id and movie_id cannot be null.
    - The rating must be within the range 0 to 5 if provided.
    - Review text and creation timestamp can be optional.

4. Users Constraints:
    - Each user must have a unique username.
    - The username cannot be null.

# Logical Database Schema

## Schema

1. Movies Table:
    - Attributes:
        - movie_id (Primary Key)
        - Title
        - Release_year
        - director_id (Foreign Key)
        - Genre
    - Attribute Descriptions:
        - movie_id: Unique identifier for each movie.
        - title: Title of the movie.
        - release_year: Year the movie was released.
        - director_id: Foreign key referencing the directors table, linking to the director of the movie.
        - genre: Genre column.
    - Default Values:
        - genre: n/a
    - Null:
        - title: Cannot be null.
        - release_year: Can be null.
        - director_id: Can be null.
        - genre: Can be null.
    - Foreign Key Action on Deletion:
        - director_id: set to null.

2. Actors Table:
    - Attributes:
        - actor_id (Primary Key)
        - First_name
        - Last_name
    - Attribute Descriptions:
        - actor_id: Unique identifier for each actor.
        - first_name: First name of the actor.
        - last_name: Last name of the actor.

- Default Values:
    - None
- Null:
    - first_name: Cannot be null.
    - last_name: Cannot be null.

3. Directors Table:
    - Attributes:
        - director_id (Primary Key)
        - First_name
        - Last_name
    - Attribute Descriptions:
        - director_id: Unique identifier for each director.
        - first_name: First name of the director.
        - last_name: Last name of the director.
    - Default Values:
        - None
    - Null:
        - first_name: Cannot be null.
        - last_name: Cannot be null.

4. Movie_Cast Table:
    - Attributes:
        - movie_id (Foreign Key)
        - actor_id (Foreign Key)
    - Attribute Descriptions:
        - movie_id: Foreign key referencing the movies table, linking to the movie in which an actor is cast.
        - actor_id: Foreign key referencing the actors table, linking to the actor who is cast in a movie.
    - Default Values:
        - None
    - Null:
        - movie_id: Cannot be null.
        - actor_id: Cannot be null.
    - Foreign Key Action on Deletion:
        - movie_id: cascade delete
        - actor_id: cascade delete

5. User_Reviews Table:
    - Attributes:
        - review_id (Primary Key)
        - user_id (Foreign Key)
        - movie_id (Foreign Key)
        - Rating
        - Review_text
        - Created_at
    - Attribute Descriptions:
        - review_id: Unique identifier for each review.
        - user_id: Foreign key referencing the users table, linking to the user who wrote the review.
        - movie_id: Foreign key referencing the movies table, linking to the movie being reviewed.
        - rating: Decimal value representing the user's rating for the movie.
        - review_text: Text containing the review written by the user.
        - created_at: Timestamp indicating when the review was created.
    - Default Values:
        - rating: 0
    - Null:
        - user_id: Cannot be null.
        - movie_id: Cannot be null.
        - rating: Can be null.
        - review_text: Can be null.
        - created_at: Can be null.
    - Foreign Key Action on Deletion:
        - user_id: cascade delete
        - movie_id: cascade delete

6. Users Table:
    - Attributes:
        - id (Primary Key)
        - Username
        - Role
        - Created_at

- Attribute Descriptions:
    - id: Unique identifier for each user.
    - username: User's username.
    - role: User's role.
    - created_at: Timestamp indicating when the user account was created.
- Default Values:
    - None
- Null:
    - username: Cannot be null.
    - role: Can be null.
    - created_at: Can be null.

SQL - (copied from VSCode)

```sql
CREATE DATABASE movies_db;

CREATE TABLE movies (
  movie_id INTEGER NOT NULL,
  title VARCHAR(255),
  release_year INTEGER,
  director_id INTEGER,
  genre VARCHAR(255),
  description TEXT,
  avg_votes DECIMAL,
  votes INTEGER

  CONSTRAINT moviesPK
            PRIMARY KEY(movie_id)
);

CREATE TABLE actors (
  actor_id INTEGER NOT NULL,
  first_name VARCHAR(255),
  last_name VARCHAR(255)

  CONSTRAINT actorsPK
            PRIMARY KEY(actor_id)
```

```
);

CREATE TABLE directors (
  director_id INTEGER NOT NULL,
  first_name VARCHAR(255),
  last_name VARCHAR(255)

  CONSTRAINT directorsPK
            PRIMARY KEY(director_id)
);

CREATE TABLE movie_cast (
  movie_id INTEGER,
  actor_id INTEGER,

    CONSTRAINT movie_cast_movieFK
              FOREIGN KEY (movie_id) REFERENCES movies(movie_id)
                        ON DELETE CASCADE ON UPDATE CASCADE,

    CONSTRAINT movie_cast_actorFK
              FOREIGN KEY (actor_id) REFERENCES actors(actor_id)
                        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE users (
  id INTEGER NOT NULL,
  username VARCHAR(255),
  role VARCHAR(255)

    CONSTRAINT usersPK
              PRIMARY KEY(id)
);

CREATE TABLE user_reviews (
  review_id INTEGER NOT NULL,
  user_id INTEGER,
  movie_id INTEGER,
  rating DECIMAL,
  review_text TEXT,
```

```
  created_at TIMESTAMP,

    CONSTRAINT user_reviews_userFK
              FOREIGN KEY (user_id) REFERENCES users(id)
                        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT user_reviews_movieFK
              FOREIGN KEY (movie_id) REFERENCES movies(movie_id)
                        ON DELETE CASCADE ON UPDATE CASCADE,

    CONSTRAINT user_reviewsPK
              PRIMARY KEY(review_id)
);
```

### Expected Operations & Volumes

The amount of transactions expected are minimal, and each transaction is linear in regards to system interaction. The volume of data is dependent on the amount of users and their reviews and the dataset chosen for the movies and their information.

1. Create
    - Create review
    - Create user
2. Retrieve
    - Retrieve movie details
3. Update
    - Update review
4. Delete
    - Delete review

## Functional Dependencies & Normalization

### Functional Dependencies

1. Movies Table:

    - movie_id -> title, release_year, director_id, genre
    - director_id -> director_name
2. Actors Table:
    - actor_id -> first_name, last_name, birthdate
3. Directors Table:
    - director_id -> first_name, last_name, birthdate
4. Movie_Cast Table:
    - (movie_id, actor_id) -> (movie_id, actor_id)
5. User_Reviews Table:
    - review_id -> user_id, movie_id, rating, review_text, created_at
6. Users Table:
    - id -> username, role, created_at

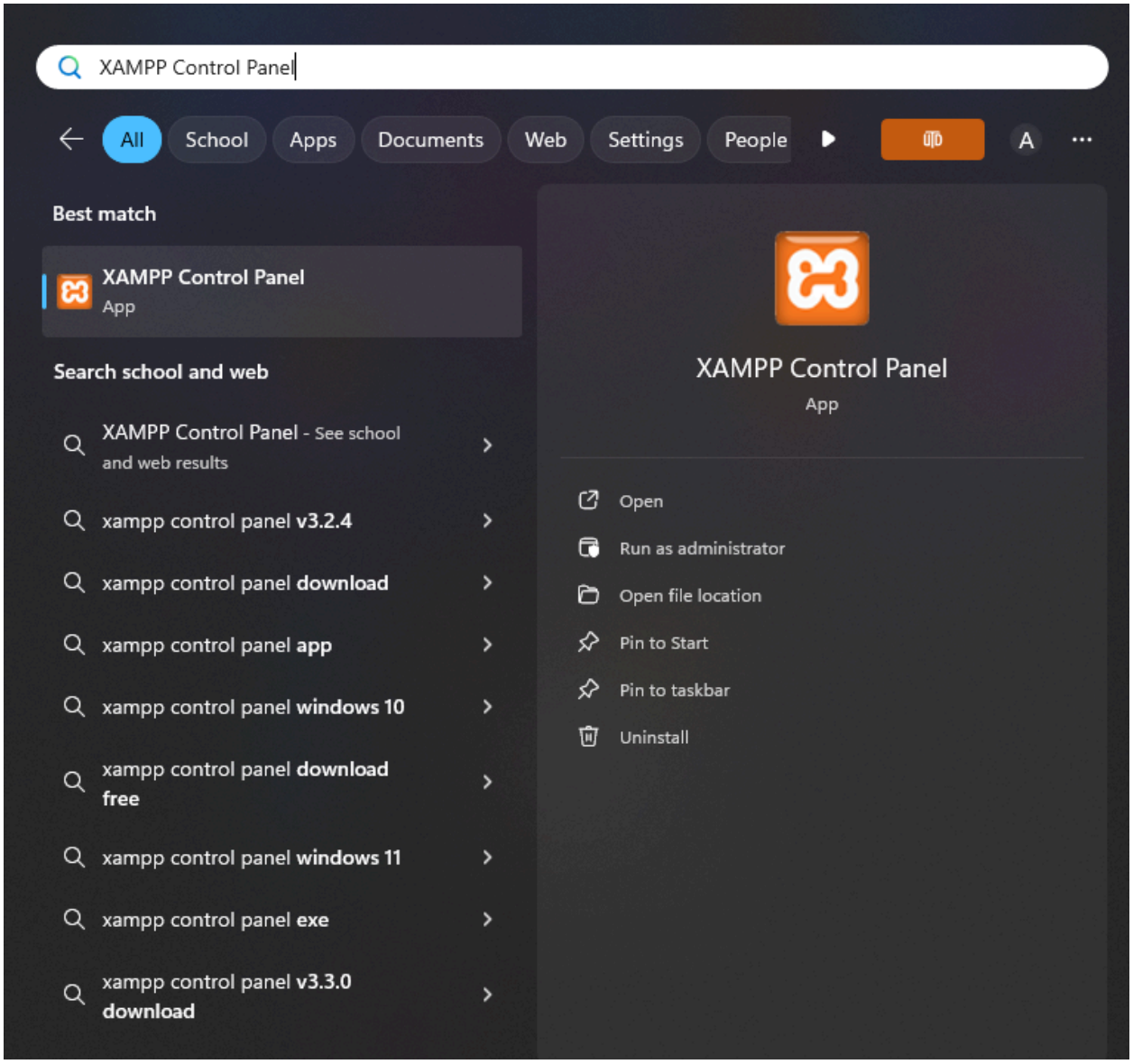
## Normalization

Database was already in 3NF.

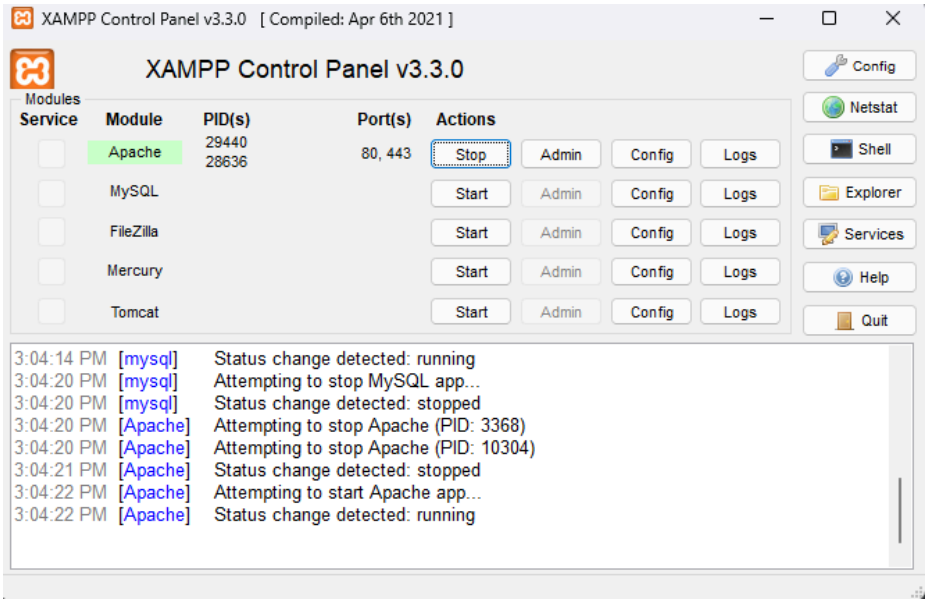
## The Database System

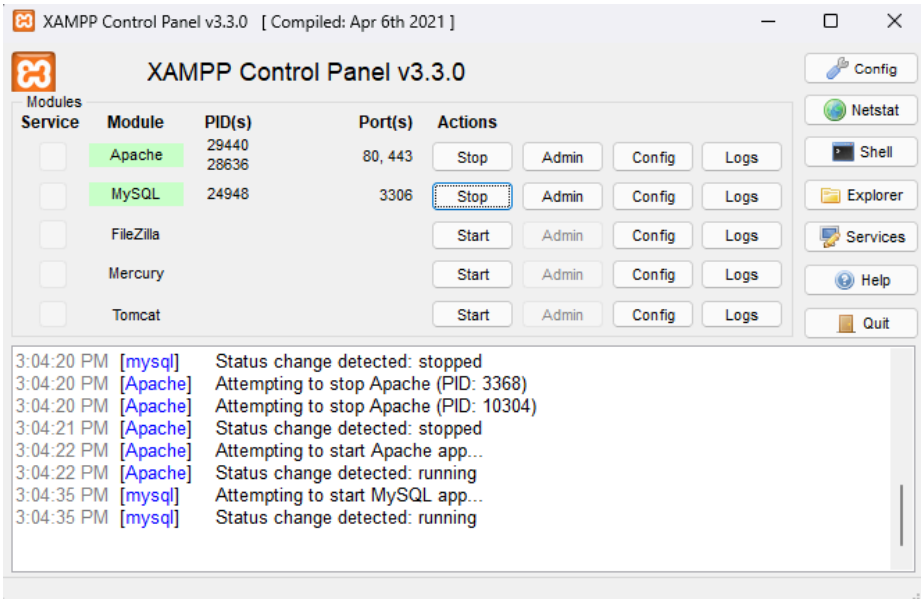
Steps:

1. Install XAMPP
    a. May need to run as administrator depending on system

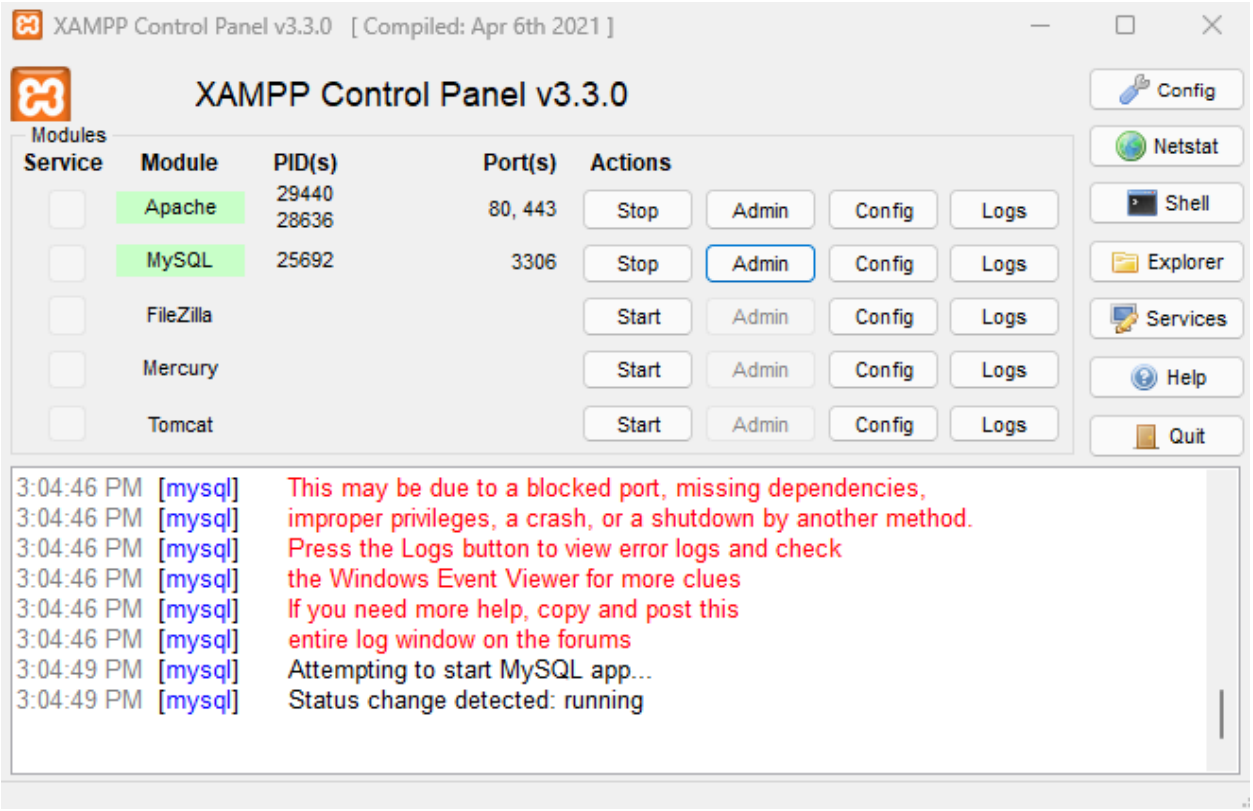


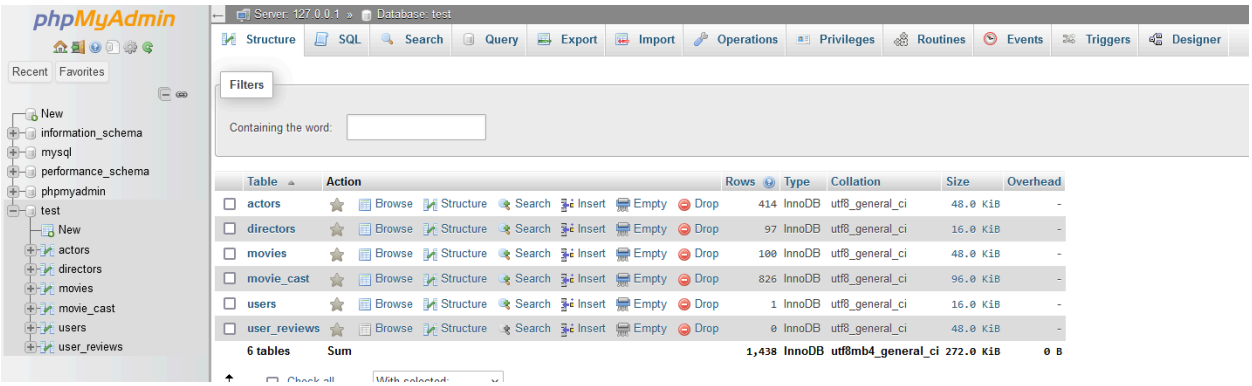
2. Install needed services & Start
    a. Apache
    b. MySQL



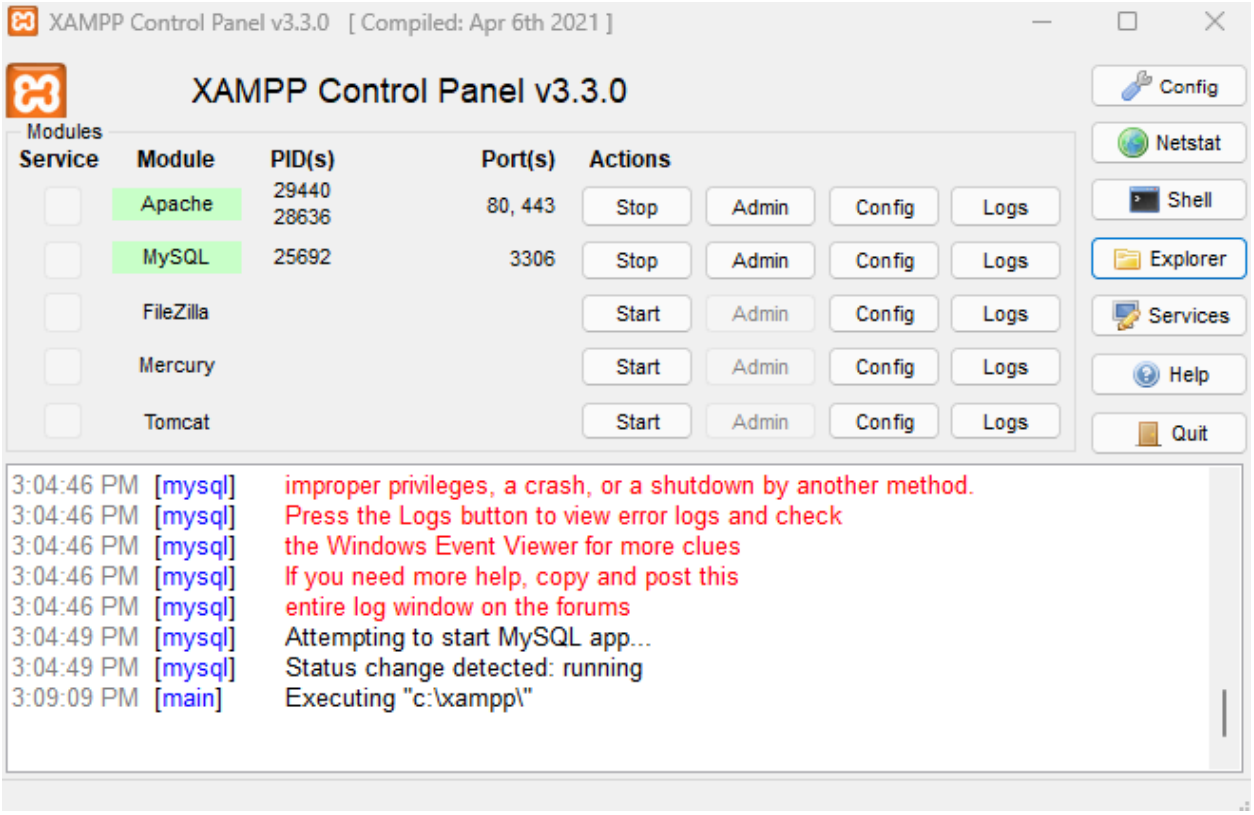



3. Click on “Admin” button to verify Apache & MySQL servers are running

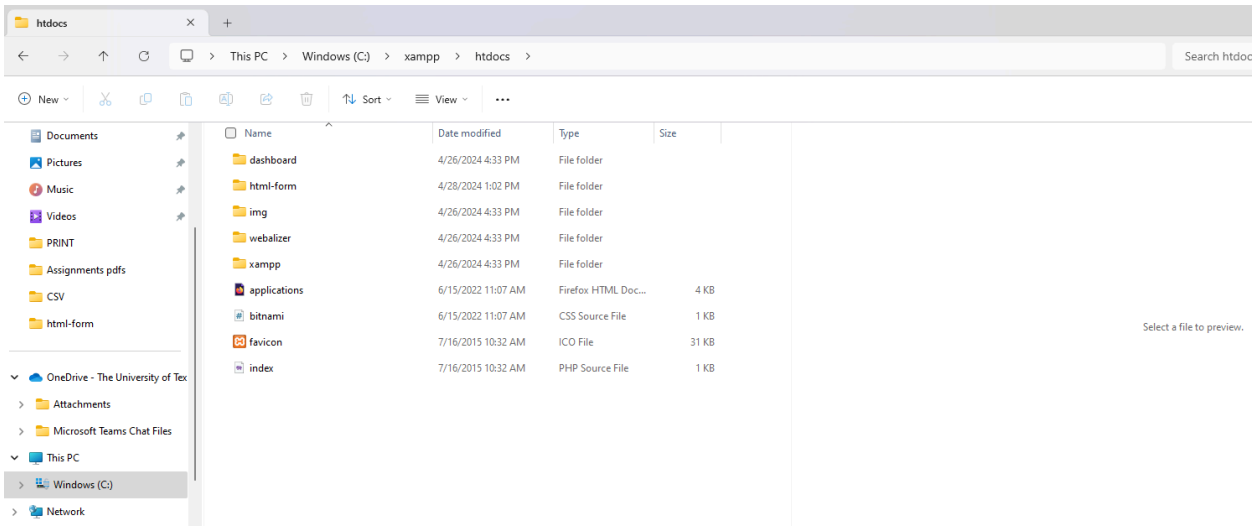


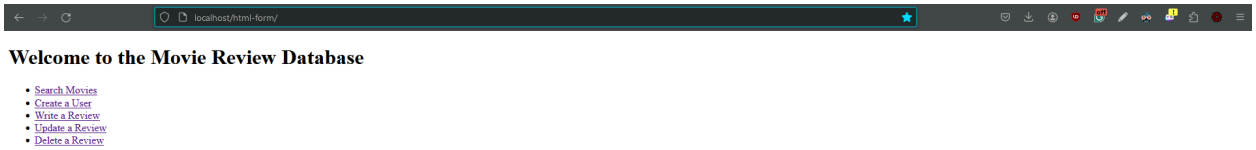



4. Click on “Explorer”

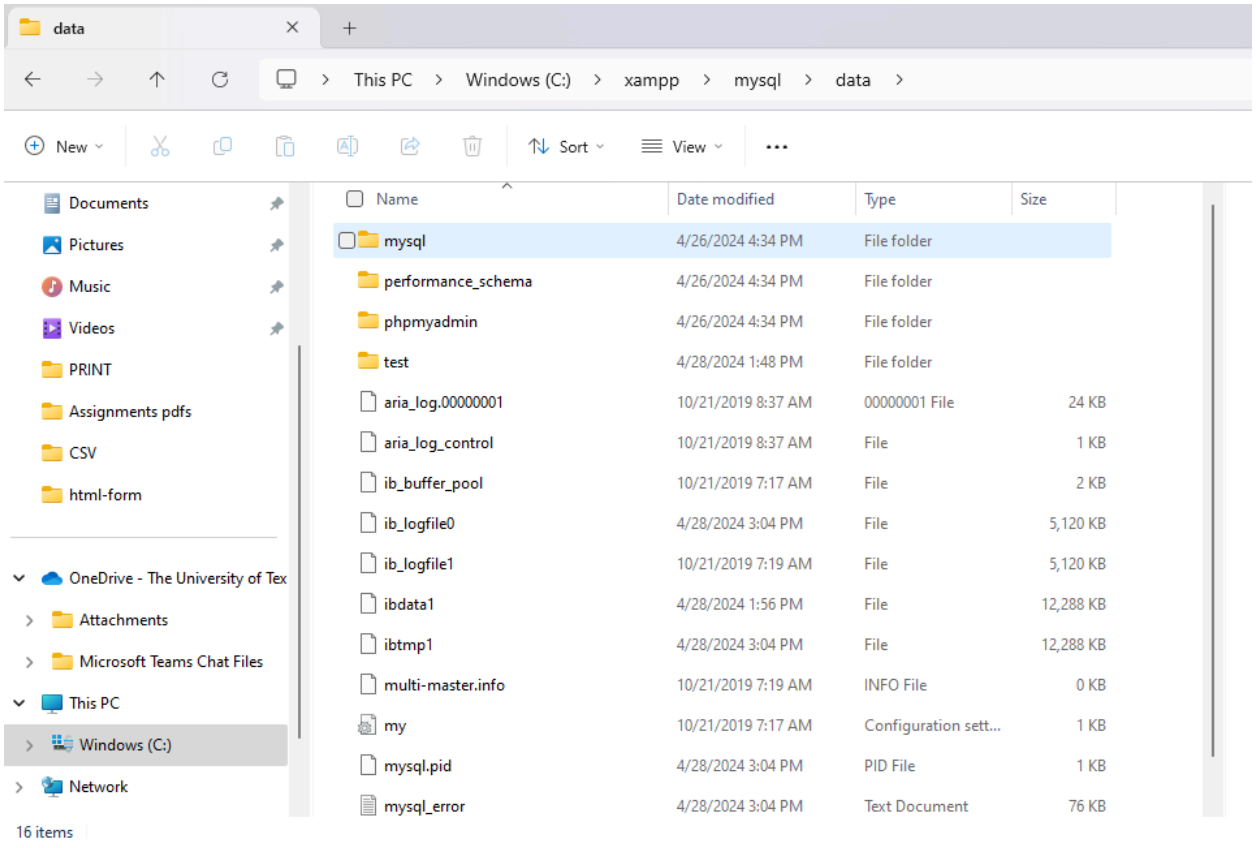



   a. Navigate to xampp > htdocs
      i. This is where Apache gets the files for hosting
      ii. Make a folder and put html and php in it
         1. “localhost/[foldername]/ is the url






b. Navigate to xampp > mysql > data > [database name]
    i. This is where any data needed for importing (.csv) is stored




After, use the website as any normal website.

## User Application Interface

Our team used HTML for creating the user interface. Users land on the index page, which displays the options for the user. It's very simple, but effective and straightforward.

An example flow:

1. User is on homepage
    a. User is presented with 5 options
        i. Search movie
        ii. Write review
        iii. Create user
        iv. Update review
        v. Delete review
2. User selects search movie
3. User inputs title or fragment of title
4. User clicks search
5. User is displayed information about movies that have a "match" or "like-match" in our database

## Conclusion & Future Work

This project was great. It gave us the freedom to complete and implement our solution on any stack we wanted (with some restrictions). It started off rough because coming up with an idea was difficult, but once we got going it became easier. Overall, it was a great experience. It was satisfying interacting with our completed project.

As for any future work, I think the main focus would be designing the website. It's plain HTML at the moment with little to no CSS or design. Another thing could be allowing users to interact with the system more. For example, user comments on reviews, uploading movies not in the database, or a user review liking/dislike functionality.

## References

https://www.apachefriends.org/docs/

https://www.w3schools.com/sql

## Appendix