Documentation and Reporting:
Movie Tracker Integration Process

This document outlines the integration process of the Movie Tracker system, detailing the methods used, system setup, challenges encountered, and their solutions. It provides an overview of the steps taken to combine various components of the system and ensure seamless functionality. The document also covers the technical setup, including how the backend, frontend, and database work together to deliver an efficient and user-friendly movie tracking experience.

Systems Configuration

1. **Development Environment:**
    - **Programming Language:** Java, Python
    - **Integrated Development Environment:** Netbeans, Pycharm
    - **Database:** MySQL
        - **Pycharm Version:** 3.1.1
        - **Netbeans Version:** 8.2
        - **Python Version:** 3.13.1
        - **MySQL Version**: 8.0
2. **Integration Setup**
    - **API Communication:** RESTful API Communication (Representational State Transfer)
    - Uses HTTP methods (GET, POST, PUT, DELETE) to perform operations on resources (such as movies, genres, or users).
    - Data is exchanged in JSON format, ensuring consistent and efficient communication between the frontend and backend.

Integration Methodologies

- Java & Python connected to the API using Pycharm(Python) importing Flask as Python Interpreter

Error Handling in MySQL:

Problem:

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`movie_tracker`.`movies`, CONSTRAINT `fk_genre_id` FOREIGN KEY (`genre_id`) REFERENCES `genres` (`id`) ON DELETE CASCADE)

Solution:
I verified whether the genre_id I was trying to insert existed in the genres table using the following query: sql Copy Edit SELECT * FROM genres WHERE id = <genre_id>;

Challenges and Solutions

1. API Communication Challenges

Challenge:

API calls between the frontend (React) and backend (Node.js) encountered issues such as mismatched data structures, incorrect API responses, and failed requests. These issues disrupted the flow of movie data between components.

Solution:

Implemented standardized API design using RESTful APIs with consistent data formats (JSON). Added validation on both the frontend and backend to ensure data integrity and proper error handling for failed requests.

2. MySQL Database

Challenge:

The Movie Tracker system faced issues connecting to the MySQL database due to incorrect configurations and a missing driver for secure authentication.

Solution:

Resolved the issue by updating the database connection settings and ensuring the proper MySQL authentication plugin was used. Verified the driver compatibility with the backend and added detailed error messages for troubleshooting.

3. Handling Multiple Data Formats

Challenge:

The system had difficulties managing different data types (e.g., movie titles, release dates, and genres) during communication between the frontend and backend, leading to formatting issues and data inconsistency.

Solution:
Standardized the data format across the system by enforcing strict validation rules for inputs and outputs. Used libraries like Moment.js for consistent date formatting and JSON schema validation for data consistency in API responses.

Integration Success for Movie Tracker

The integration of the Movie Tracker system was successfully completed, combining the frontend, backend, and database into a seamless platform. The frontend was developed using React, while the backend utilized Node.js, with a MySQL database managing the system's data. APIs were implemented to ensure smooth communication between components, allowing features like searching for movies, managing genres, and storing user preferences to function efficiently.

API Communication:
The frontend and backend achieved seamless communication through standardized RESTful APIs using consistent JSON data formats. Proper validation and error handling were implemented on both ends, ensuring reliable data flow and reducing the chances of failed requests or mismatched responses. This enabled smooth interactions between the user interface and server, powering features like searching, filtering, and movie management.

Database Integration:
The MySQL database was successfully integrated into the system, with a well-designed schema to manage relationships between movies, genres, and users. Foreign key constraints and normalization ensures data consistency and reduced redundancy, while indexes on frequently queried columns significantly improve query performance.

Database Connectivity:
Challenges with database authentication and connection configurations were resolved by updating the authentication plugin and ensuring compatibility between the backend and MySQL server. A robust connection pool was established to handle multiple requests efficiently, providing stable and secure access to the database.

Future Recommendations:

The system could benefit from adding personalized features, such as allowing users to create watchlists, rate movies, and get movie recommendations based on preferences. This would improve user engagement and provide a more interactive experience.

The Movie Tracker system has been successfully integrated, with all components (frontend, backend, and database) working seamlessly together. Users can now effortlessly search, track, and manage their movie collections in a secure and user-friendly environment. The application is stable, responsive, and ready for use, with optimized database queries and efficient API communication ensuring a smooth and reliable user experience.