MovieMall Web Application

.

General

- #### Team#:
- #### Names:
- #### Project 5 Video Demo Link:
- #### Instruction of deployment:
- #### Collaborations and Work Distribution:

•

Connection Pooling

- #### Include the filename/path of all code/configuration files in GitHub of using JDBC Connection Pooling.
- #### Explain how Connection Pooling is utilized in the Fabflix code.
- #### Explain how Connection Pooling works with two backend SQL.

•

Master/Slave

- #### Include the filename/path of all code/configuration files in GitHub of routing queries to Master/Slave SQL.
- #### How read/write requests were routed to Master/Slave SQL?

•

JMeter TS/TJ Time Logs

- #### Instructions of how to use the `log_processing.*` script to process the JMeter logs.

JMeter TS/TJ Time Measurement Report

| Single-instance Version Test Plan | Graph Results Screenshot | Average Query Time(ms) | Average Search Servlet Time(ms) | Average JDBC Time(ms) | Analysis | |----------|--| | Case 1: HTTP/1 thread | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? | | Case 2: HTTP/10 threads | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? | | Case 3: HTTPS/10 threads | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? | | Case 4: HTTP/10 threads/No connection pooling | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? |

| Scaled Version Test Plan | Graph Results Screenshot | Average Query Time(ms) | Average Search Servlet Time(ms) | Average JDBC Time(ms) | Analysis | |----------|--| | Case 1: HTTP/1 thread | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? | | Case 2: HTTP/10 threads | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? | | Case 3: HTTP/10 threads/No connection pooling | to image in img/.png to image in img/.pdf to image in img/.jpg to image in img/.jpeg to image in img/.bmp to image in img/.tiff to image in img/.tif to image in img/.gif to image in img/.eps to image in img/.ps to image in img/.eps.gz to image in img/.ps.gz to image in img/.eps.Z | ?? | ?? | ?? | ?? |

Click here to visit the MovieMall website! Click here to visit the MovieMall website!

Click here to watch the demo!

Table of Contents

- Contributors

Overview

MovieMall is a meticulously crafted web application designed to offer movie and star details with an intuitive and responsive interface. We've taken special care to ensure a clear separation between frontend and backend components, guaranteeing modularity and ease of future updates.

Û Backend Services

Our robust backend leverages Servlets to provide endpoints for extracting comprehensive information about movies and stars.

Technology Stack:

- Java: 19.0.2
- Apache Maven: 3.8.7
- Apache Tomcat: 10.1.13
- Jakarta Servlet: 6.0.0
- JSON: For data interchange
- JDBC: For seamless database connectivity

Main Components:

1. Servlets: Expertly manage HTTP requests and deliver data in a crisp JSON format.
2. Database Manager: Orchestrates database connections and spearheads SQL query executions.
3. Adapters: Act as the bridge, transforming data from database result sets to Java objects and facilitating JSON conversions.
4. XML Parsings: Parsing multiple xml and inserting them into the database according to rules.

Performance Optimization Strategies

This section details the optimization strategies implemented in our XML parsing project. A key highlight is the use of multi-threading, which is intricately designed to enhance processing speed and efficiency.

1. Multi-Threading Implementation

   - Thread Pool and Executors: We utilize `Executors.newFixedThreadPool(3)` to create a pool of threads, which allows for parallel processing of multiple XML files. This approach significantly reduces the time required for parsing large XML files as compared to a sequential approach.
   - Concurrent HashMaps: `ConcurrentHashMap` is used for `sharedStarMap` and `sharedMovieMap`, ensuring thread-safe operations while allowing concurrent reads and updates. This is crucial in a multi-threaded environment to prevent data corruption.
   - CountDownLatch Mechanism: The `CountDownLatch` is used to synchronize the completion of tasks. For instance, after parsing movies and stars in separate threads, a latch ensures both are completed before initiating the cast parsing. This ensures data consistency and integrity.

2. Efficient Data Structures

   - Custom Data Structures: Depending on the specific access patterns observed during parsing, such as more frequent reads than writes, custom data structures with read-write locks can be used to improve performance.
   - Data Access Optimization: By analyzing how data is accessed and modified in the maps, we can reduce contention and improve throughput. For instance, using temporary local structures to accumulate data before a single batch update to the shared map can minimize lock contention.

3. Batch Processing for Database Operations

   - Batch Database Operations: Instead of individual inserts or updates for each record, we implement batch operations. This approach reduces the number of network calls and database I/O operations, significantly decreasing the total execution time.
   - Bulk Insertions: Where possible, we leverage bulk insertion techniques provided by the database to handle large data sets more efficiently. This reduces the overhead associated with individual row insertions.

Servlet Endpoints:

- MovieListServlet: Lists the crème de la crème of movies.
  - Path: /MovieListServlet
  - Method: GET
  - Response: A curated JSON array of top 20 movies, ranked by ratings.
- MovieDetailServlet: Delve deeper into the intricacies of a particular movie.
  - Path: /MovieDetailServlet
  - Method: GET
  - Parameters: query - A Base64 encoded movie ID.
  - Response: A detailed JSON depiction of a movie.
  - Error Handling: Gracefully handles situations where a movie isn't found or the URL encounters issues.

- StarDetailServlet: Unveil details of a shining star.
  - Path: /StarDetailServlet
  - Method: GET
  - Parameters: query - A Base64 encoded star ID.
  - Response: A rich JSON portrayal of a star and their cinematic journey.

Detailed Backend Features:

- Database Operations: All interactions with the database are managed via the DatabaseManager class.
- Data Adapters: These are pivotal in molding database results into domain-specific entities and then crafting them into JSON.
- Utility Functions: Classes like `URLUtils` augment functionality, especially in URL parameter handling.
- Exception Handling: Our built-in ExceptionHandler guarantees consistent error responses, ensuring transparency with the client.
-

Frontend Details

Technology Stack:

- React.js: 18.2.0
- HTML: 5
- CSS: 3
- JavaScript: Enhanced with AJAX for asynchronous data retrieval
- Node.js: 20.5.0
- npm: 9.8.0
- Bootstrap: For a fluid, responsive design

Й Features:

1. Search Bar: Dive into a world of movies and stars.
2. Listing Page: A showcase of top-tier movies, complete with pagination.
3. Detail Page: A closer look at your favorite movie or star.

Components:

- Navbar: A handy navigation tool for a seamless browsing experience.

Setup and Installation

1. Ensure that Java 19 or higher is installed on your system.
2. Clone the repository.
3. Navigate to the moviemall-server directory and use the following command, Apache Maven will download the dependencies required by the project. ```bash mvn clean install
4. After that, use the following command to package it into a WAR file. ```bash mvn package
5. Adjust your database settings.
6. In the moviemall-client directory, use npm to build the React project. This will generate a static directory. ```bash npm run build
7. Move the static content generated from the React build into the appropriate location within the WAR directory structure.
8. Deploy the combined WAR file to a servlet container, like Tomcat.
9. Access the application using your preferred web browser via the server's address.

APIs

- GET `/MovieListServlet`: Spotlight on top-rated movies.
- GET `/MovieDetailServlet?query=<movie_id>`: A cinematic deep dive into a specific movie.
- GET `/StarDetailServlet?query=<star_id>`: Illuminate the life of a star.

Contributors

- Jiahao Liangắ
  - Utilize React to modularize the web servlet, segmenting the application based on specific functionalities.
  - Develop the frontend code to enhance the website's aesthetic appeal.
  - Create the demo video.
- Xiaohua Zhangắ
  - Implement the web servlet and draft SQL queries for backend logic.
  - Conduct comprehensive website testing to ensure optimal functionality and the satisfaction of project requirement.