

Project 4 – pratyusj

Name: Pratyush Jain; Email: pratyusj@andrew.cmu.edu

Walkthrough(a narrated screencast that includes the same information that would be in the writeup.): <https://youtu.be/8zqges5eskQ>

If this doesn't work, please let me know at pratyusj@andrew.cmu.edu

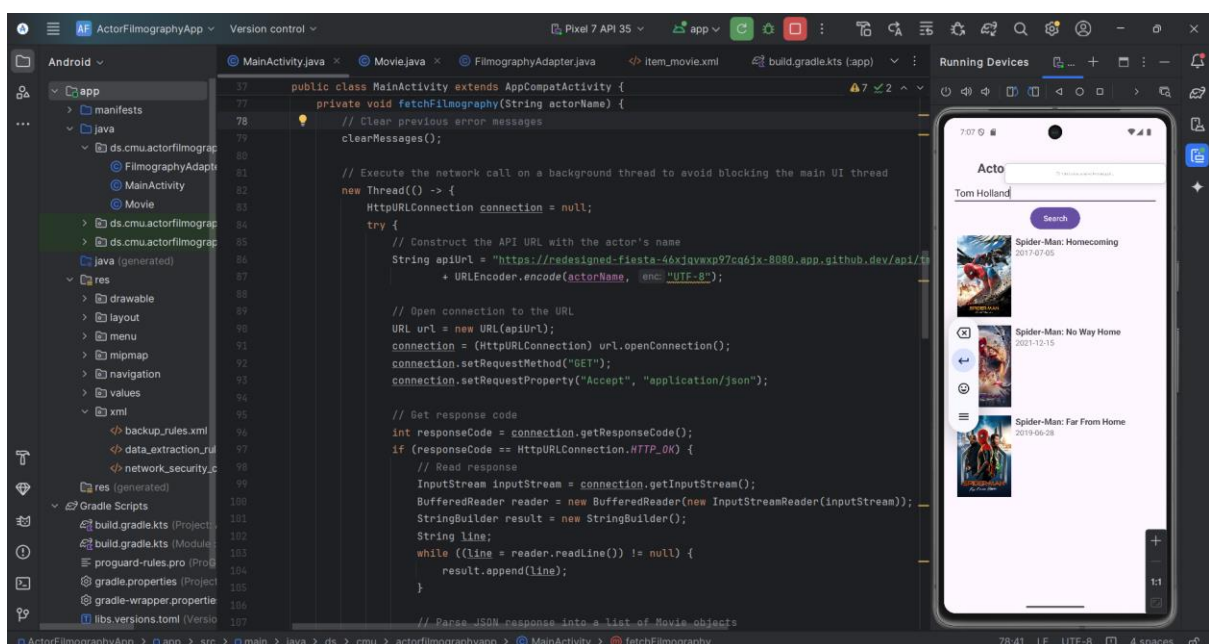
API: The Movie DB (www.themoviedb.org)

API Documentation: [Getting Started](https://developer.themoviedb.org/docs/getting-started) (<https://developer.themoviedb.org/docs/getting-started>)

Short description: My application will allow users to enter a movie title and retrieve important information about the movie, like its release date, rating, etc.... using the TMDb API. This data will help users quickly access movie details without needing to search through multiple platforms.

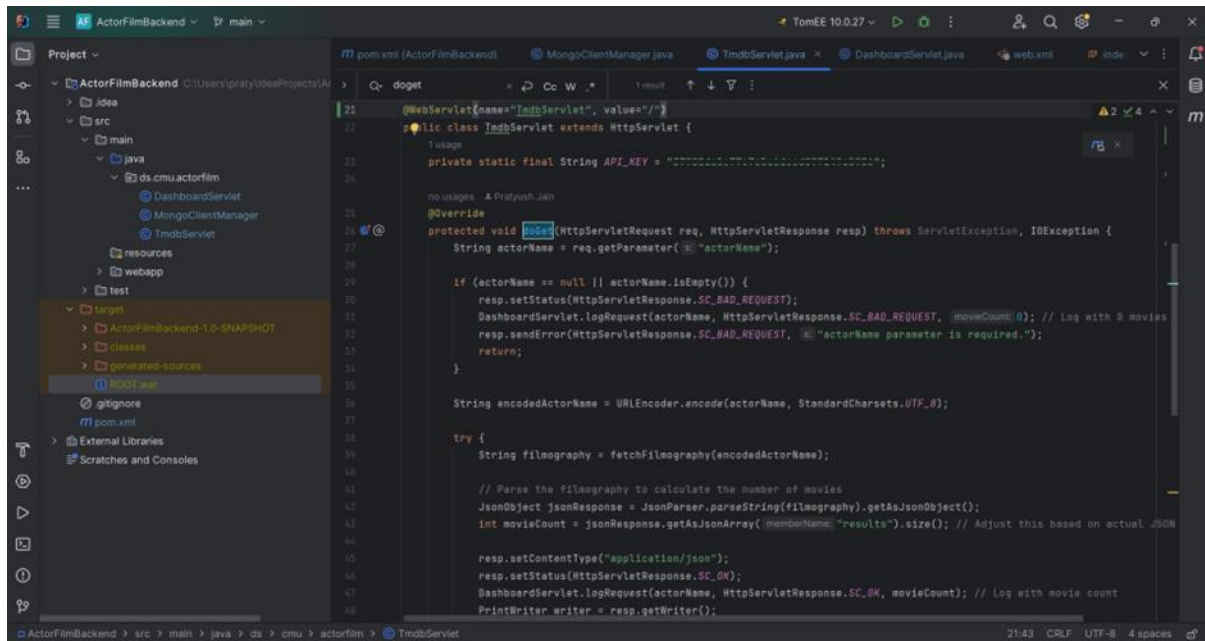
1. Implement a native Android application

The Android app allows users to search for my favourite actor(s) and recommend 3 movie. It has a clean UI with TextView, EditText, Button, and RecyclerView components. Users can enter an actor's name, and the app sends an HTTP request to the backend service, parses the JSON response, and displays movie details like title, release date, and posters. The app handles errors gracefully, eg. displaying a "No movies found" message when the search gives no results.



2. Implement a web service

The backend web service, written in Java using Servlets and hosted on TomEE-plus, and later on codespaces, it handles requests from the Android app. It accepts an actor's name via an HTTP GET request, queries the TMDb API for film details, and returns the results in JSON format. The service processes third-party API responses, logs each request, and ensures that data is consistent before sending response back to app.



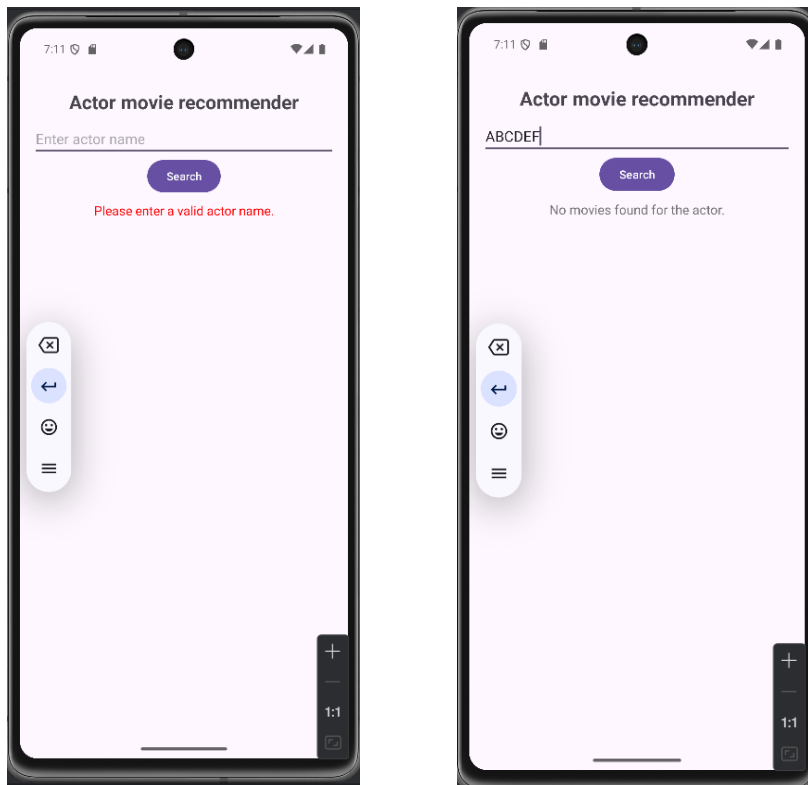
The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure for 'ActorFilmBackend'. The 'src' directory contains 'main' and 'test' folders. The 'main' folder contains a 'java' directory with a package 'ds.cmu.actorfilm'. Inside this package are files for 'DashboardServlet', 'MongoClientManager', 'TmdbServlet', 'resources', 'webapp', and 'test'. The 'target' directory contains 'ActorFilmBackend-1.0-SNAPSHOT', 'classes', 'generated-sources', and 'javadoc'. The 'pom.xml' file is also visible.
- Editor:** Displays the code for 'TmdbServlet.java'. The code is as follows:

```
21 @WebServlet(name="TmdbServlet", value="/")
22 public class TmdbServlet extends HttpServlet {
23     1 usage
24     private static final String API_KEY = "00000000000000000000000000000000";
25
26     no usages. A Pratyush Jain
27     @Override
28     protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
29         String actorName = req.getParameter("actorName");
30
31         if (actorName == null || actorName.isEmpty()) {
32             resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
33             DashboardServlet.logRequest(actorName, HttpServletResponse.SC_BAD_REQUEST, movieCount); // Log with 0 movies
34             resp.sendError(HttpServletResponse.SC_BAD_REQUEST, "actorName parameter is required.");
35             return;
36         }
37
38         String encodedActorName = URLEncoder.encode(actorName, StandardCharsets.UTF_8);
39
40         try {
41             String filmography = fetchFilmography(encodedActorName);
42
43             // Parse the filmography to calculate the number of movies
44             JSONObject jsonResponse = JsonParser.parseString(filmography).getAsJsonObject();
45             int movieCount = jsonResponse.getAsJsonArray("results").size(); // Adjust this based on actual JSON
46
47             resp.setContentType("application/json");
48             resp.setStatus(HttpServletResponse.SC_OK);
49             DashboardServlet.logRequest(actorName, HttpServletResponse.SC_OK, movieCount); // Log with movie count
50             PrintWriter writer = resp.getWriter();
```

3. Handle error conditions

Error handling is implemented both in the app and backend. The app validates user input and displays appropriate messages for network issues or server errors. On the server, invalid actor names result in a 400 Bad Request, and failures in communicating with TMDb return a 500 Internal Server Error. Unexpected responses from TMDb are logged, and users receive meaningful error messages. It also filters out info like: missing name, no actors./movies found/bad API call, etc.



4. Log useful information

My backend logs six pieces of data for each request: the actor's name, HTTP status code, number of movies/pages returned, timestamp, and whether the request originated from the app. This information is crucial for debugging and analytics, enabling detailed tracking of app usage. Also we can see all logs below that as requested.



API Request Logs Dashboard

Dashboard Analytics

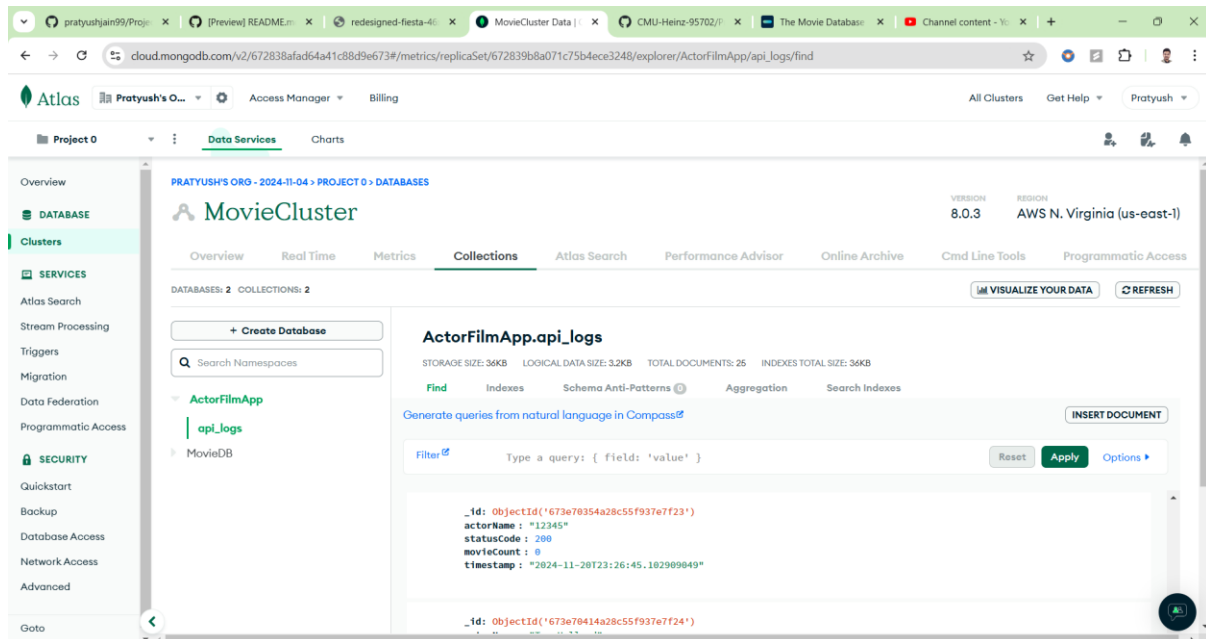
- Total Requests: 17
- Successful Requests: 17
- Most Searched Actor: Benedict Cumberbatch (10 searches)
- Average Number of Pages (of Movies) Returned: 1.3529411764705883
- Unique Actors Searched: 6
- Success Rate: 100.00%

Logs

#	Actor Name	Status Code	Pages(of movies) Returned	Timestamp
1	Benedict Cumberbatch	200	1	2024-11-20T16:23:50.324318800
2	Benedict Cumberbatch	200	1	2024-11-20T16:23:52.501638300
3	Benedict Cumberbatch	200	1	2024-11-20T16:23:53.199166500
4	Benedict Cumberbatch	200	1	2024-11-20T16:23:54.439152900
5	Benedict Cumberbatch	200	1	2024-11-20T16:23:55.520120900
6	Tom Hiddleston	200	1	2024-11-20T16:24:05.909243500
7	Robert Downey Jr	200	1	2024-11-20T16:24:14.899180500
8	Benedict Cumberbatch	200	1	2024-11-20T21:25:08.327478246
9	Tom Hiddleston	200	1	2024-11-20T16:39:07.653797200
10	Tom Hanks	200	2	2024-11-20T16:39:21.231206200
11	Benedict Cumberbatch	200	1	2024-11-20T21:49:48.990771298
12	Benedict Cumberbatch	200	1	2024-11-20T22:06:31.505631215
13	Benedict Cumberbatch	200	1	2024-11-20T23:22:04.987811738
14	Benedict Cumberbatch	200	1	2024-11-20T23:22:49.220796472
15	Tom Hanks	200	2	2024-11-20T23:26:28.118407297
16	12345	200	0	2024-11-20T23:26:45.102909049
17	Tom Holland	200	6	2024-11-20T23:26:57.557829691

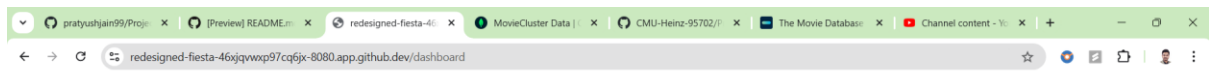
5. Store the log information in a database

This is snapshot from my MongoDB atlas page, Logs are stored persistently in MongoDB Atlas. Each log entry includes the actor's name, status code, movie count, and timestamp. MongoDB's Java Driver is used to perform CRUD (no deletions actually) operations, ensuring that all logs are efficiently stored and accessible for future analysis.



6. Display operations analytics and full logs on a web-based dashboard

The dashboard is a web-based interface built using JSP. It displays analytics such as total requests, successful requests, the most searched actor, the average number of movies returned, unique actors searched, and success rate. Logs are presented in a table format, showing detailed information for each request, including timestamps and results. This is again followed by all the logs! (on next page)



API Request Logs Dashboard

Dashboard Analytics

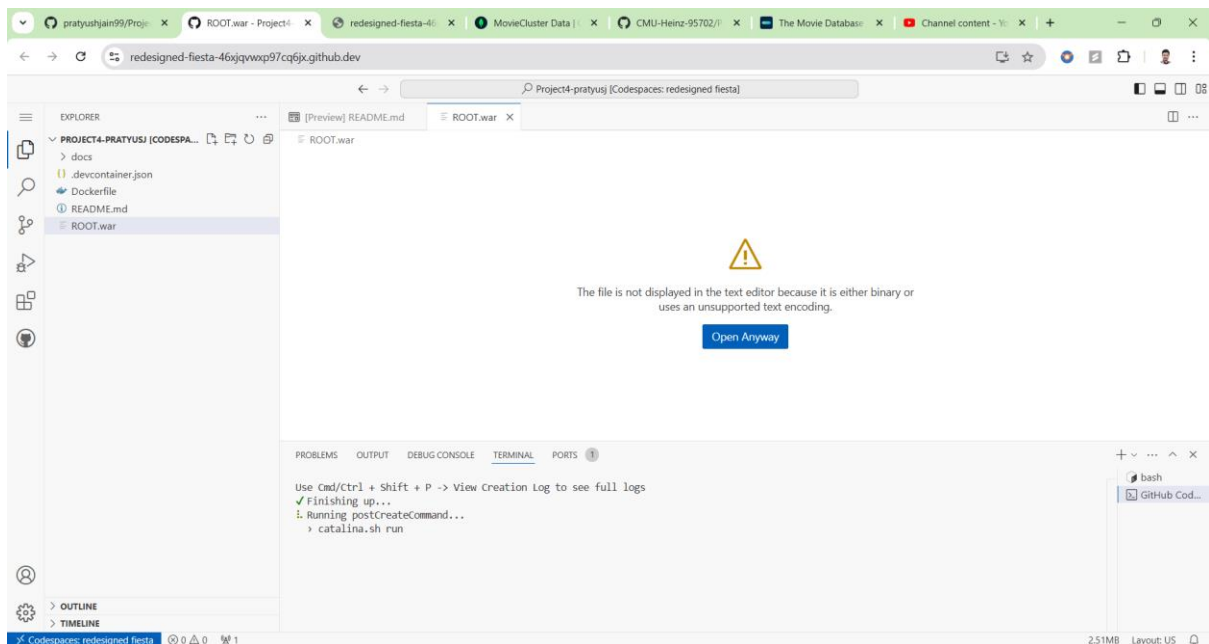
- Total Requests: 17
- Successful Requests: 17
- Most Searched Actor: Benedict Cumberbatch (10 searches)
- Average Number of Pages (of Movies) Returned: 1.3529411764705883
- Unique Actors Searched: 6
- Success Rate: 100.00%

Logs

#	Actor Name	Status Code	Pages(of movies) Returned	Timestamp
1	Benedict Cumberbatch	200	1	2024-11-20T16:23:50.324318800
2	Benedict Cumberbatch	200	1	2024-11-20T16:23:52.501638300
3	Benedict Cumberbatch	200	1	2024-11-20T16:23:53.199166500
4	Benedict Cumberbatch	200	1	2024-11-20T16:23:54.439152900
5	Benedict Cumberbatch	200	1	2024-11-20T16:23:55.520120900
6	Tom Hiddleston	200	1	2024-11-20T16:24:05.909243500
7	Robert Downey Jr	200	1	2024-11-20T16:24:14.899180500
8	Benedict Cumberbatch	200	1	2024-11-20T21:25:08.327478246
9	Tom Hiddleston	200	1	2024-11-20T16:39:07.653797200
10	Tom Hanks	200	2	2024-11-20T16:39:21.231206200
11	Benedict Cumberbatch	200	1	2024-11-20T21:49:48.990771298
12	Benedict Cumberbatch	200	1	2024-11-20T22:06:31.505631215
13	Benedict Cumberbatch	200	1	2024-11-20T23:22:04.987811738
14	Benedict Cumberbatch	200	1	2024-11-20T23:22:49.220796472
15	Tom Hanks	200	2	2024-11-20T23:26:28.118407297
16	12345	200	0	2024-11-20T23:26:45.102909049
17	Tom Holland	200	6	2024-11-20T23:26:57.557829691

7. Deploy the web service to GitHub Codespaces

Deployment was tested for successful communication between the app and the backend, with thorough checks for error scenarios and API reliability.



Walkthrough (a narrated screencast that includes the same information that would be in the writeup.): <https://youtu.be/8zqges5eskQ>

Initial Github deployment at: <https://github.com/pratyushjain99/Project4-pratyusj>