Test Strategy :

1. **Test Plan/Suites:** Create test suite or test plan for each module and test cases based on user Acceptance criteria. So it will help to cross check the test coverage.
2. **Module Allocation (API Suits) :** Based on team strength, assign modules to each team member and allocate the time. Module allocation helps to check the test coverage and yet not overlapping.
3. **Testing Techniques/Tools:** Based on project requirement/time and knowledge, manual or automated tests or combinations are defined. Manual testing should be preferred for the very first testing of any application.
4. **Test Schedule:** Time estimation should be based on how many test cases contained in each module. If testing tools are defined then firstly, high priority test cases should be scripted and run. Time estimation contains new, retest and regression tests.
5. **Prioritization:** After assigning modules and estimating time, have to prioritize the modules before starting testing. It should be high, medium and low.
6. **Test Reports:** Based on each test report, we can conclude how many modules(API suits) are tested.

Test Types :

1. **Functional API Tests -**
    - Positive test case - It is also called a happy path. It is basically used to test basic functionality and acceptance criteria of API.
    - Negative test case - Here we handle the problem scenario with invalid user and invalid inputs.
2. **Performance API Tests -**
    - Load Test - Here we basically use virtual users, time period and the specific API.
    - Stress Test - Here we start with low or medium users and then continuously increase the number of users to check which API slows down or becomes non responsive.
    - Soak Test - This test basically performs for a high number of users for a long time to check whether it performs the same for a longer time or not.
    - Spike Test and Peak Test - Here, we can suddenly increase the load for a shorter time to check the API scalability. It helps in auto scaling infrastructure.
    - Regularly or periodically API monitoring also helps to find out the potential problems with the system in production. Based on the problem, the user is able to optimise selected API.
3. **Security API Tests -** It can be achieved by using invalid input, missing/unauthorized token, non existing endpoints and assert that API provide valid response or error handling and doesn't break.

Test Actions :

1. **Verify Correct HTTP status code** : E.g. 200,201 for all success code, 400, 403 for bad request, 500 - server error, 504 - server timeout, 404 - Not found
2. **Verify response payload :** Check valid JSON body and correct field names, types, and values
3. **Verify response headers :** HTTP server headers have connection on both security and performance.

Test Flow :

1. **Testing requests in isolation** – Executing a single API request and checking the response accordingly.

2. **Multi-step workflow with several requests** – Testing a series of requests which are common user actions, since some requests can rely on other ones.

Test scenario based on given API endpoints -

1. Validate all API calls return valid status code in ***positive tests***:
    - All requests should return 2XX HTTP status code
    - Returned status code is according to spec:
        – 200 OK for GET requests
        – 201 for POST or PUT requests creating a new resource
        – 200, 202, or 204 for a DELETE operation
2. Validate payload all API calls with valid :
    - Response is a well-formed JSON object
    - Response structure is according to data model like schema validation: field **names** and field **types** are as expected

3. Validate state like GET, PUT, POST, DELETE - e.g GET request has no state change option
4. Validate headers :
    - Verify that HTTP headers are as expected, including - content-type, connection, cache-control, expires, access-control-allow-origin, keep-alive
5. Verify the response of any API is received in a timely manner. (test plan)
6. Verify all responses should be valid and based on status code.
7. Validate all API calls return valid status code in ***negative tests with valid input.***
    - Attempt to perform post/put/delete by using user's login
    - Attempt to add duplicate id in PUT method
    - Attempt to delete non-existing id
    - Attempt to add invalid value like imdb score or value more than the greater range

8. Validate all API calls return valid status code in ***negative tests with invalid input***:
    - Missing or invalid authorization token
    - Invalid value in HTTP header
    - Invalid/missing API endpoints
    - Missing/Invalid fields in payload
    - Invalid id/uuid in parameter
    - Invalid content-type in payload
    - Overflow payload - huge JSON body
    - Attempt to get data by user with expired access / inactive user
    - Attempt/Invalid query parameter
9. Validate versioning of old and new API.

Tool/Software - There are many tools available in the market but based on my experience Postman and Jmeter are the finest tools for API and load testing.

Load Balancing -

1. Distribution of work over several servers - Load Balancing
2. Case 1- If multiple requests hit a single api, then we can pass some proportion of requests to another server.
3. Case 2 - When server X is reached upto defined threshold value, then the next all request directly shifts to server Y and so on.
4. Case 3 - Send a new request to the server with the fewest current connections.
5. Case 4 - Send a new request to the server with the fastest response time and fewest active connections
6. Since it's all static data so we can use caching instead of DB query to speed up response.
7. We can use Database indexing to speed up response. So on every GET api call, indexing can be used to quick response instead of searching every row for results.

Improvement in System -

1. Filter movie by date/release date.
2. Filter list by its type like - new/old/bollywood/hollywood
3. Trending top 10 list features.
4. Recommend list based on watched movie
5. If applied filter by top scored movie then already watched movie should not be included in the list.
6. Add pagination for large data.
7. We can use user authorization to save user details so users can add a watchlist, save to their favourite, shared movie, add reminder.
8. Add search feature - Top 10 search, search by using movie name, genre
9. Add movie description, movie time, rate movie, cast, maturity rating.
10. Recommendation based on multiple languages.

Test Architecture -

Test architecture is showing how the different modules of the system communicate with each other and other systems.

In this project, admin and user both have API access but based on its authorization. So admin can perform a little more functionality than normal users like add/update/delete movie data. Where normal users only can see the movie list and selected movie with details.

Based on user role below are some user actions listed -

User Actions -

- **Admin -**
    1. Admin can use GET/movies api to see all movie lists.
    2. Admin can use GET /movies<id> api to see selected movie.
    3. Admin can use PUT/movies api to modify selected field.
    4. Admin can use POST/movie api to add new movies to the database.
    5. Admin can use DEL/movie/<id> api to delete any movie.
- **User -**
    1. User login can use GET/movies api to see all movie lists.
    2. User login can use GET /movies<id> api to see selected movie.
    3. If a user login tries to use PUT, POST or DELETE api then valid response code and message should be displayed. (validate authorization)

- **Common in both -**
    1. If the admin deletes any movie then on GET api valid response should be displayed.
    2. When the admin adds a new movie list using POST api then in the next GET api call, an updated list should be displayed.
    3. When the admin performs PUT operation then modified data should be displayed in the next GET api call.