Queen Mary University of London, MSc Big Data Science 2021-22

Cloud Computing Mini Project

Project Specification Document - Group 12

This document specifies project requirements, scope, application design and features.

Table of Contents

[Version History 3](#_Toc66476353)

[1. Objective 4](#_Toc66476354)

[2. Specifications 4](#_Toc66476355)

[2.1. Endpoints 4](#_Toc66476356)

[2.1.1. /browse 4](#_Toc66476357)

[2.1.2. /signup 4](#_Toc66476358)

[2.1.3. /login 4](#_Toc66476359)

[2.1.4. user/createlist 5](#_Toc66476360)

[2.1.5. user/addtolist 5](#_Toc66476361)

[2.1.6. user/deletefromlist 5](#_Toc66476362)

[2.1.7. user/deletelist 5](#_Toc66476363)

[2.1.8. /deleteuser 6](#_Toc66476364)

[2.1.9. user/logout 6](#_Toc66476365)

[2.2. Database schema 6](#_Toc66476366)

[2.2.1. Table 1 – Username and lists 6](#_Toc66476367)

[2.2.2. Table 2 – Username and session id 6](#_Toc66476368)

[2.2.3. Table 3 – Titles 7](#_Toc66476369)

[3. Testing 8](#_Toc66476370)

[4. Deployment 8](#_Toc66476371)

[5. Collaboration and Maintenance 8](#_Toc66476372)

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Version id | Date | Author | Description |
| 1.0 | 12/03/2021 | Abhinav Jain | Initial version |
|  |  |  |  |
|  |  |  |  |

# Objective

This project aims to build a backend application with RESTful architecture. The project will create an application to browse TV show titles and save personalized list of TV titles. The application will follow the REST standard and will be hosted on a cloud platform.

# Specifications

# Endpoints

# /browse

* This endpoint will implement GET method.
* The request body must contain a TV title user wants to browse.
* If request body doesn’t have a TV title, an error message with suitable error code will be returned.
* If TV title is present in the request, call external API <http://api.tvmaze.com/search/shows?q=<enter> TV title here>. If API returns the data with success message code, relay the output as the output of this endpoint.
* If above API call results in error code, return an error message with suitable error code.

# /signup

* + - This endpoint will implement a POST method.
    - The request body must contain a username and a password.
    - If username or password is missing from the request, return an error message with suitable code.
    - If username already exists, return an error message with suitable code.
    - If no error is found, create the username and password and save in the database. Password must be hashed before saving in the database. Return a success message with suitable code and username in the response.

# /login

* + - This endpoint will implement a POST method.
    - The request body must contain a username and a password.
    - If username or password is missing from the request, return an error message with suitable code.
    - Username and password (hashed) combination must be present in the database, if not, an error message must be returned with suitable error code. If a match is found success message should be returned with suitable code and session id. Session id should be saved in database as active session id.

# user/createlist

* + - This endpoint will implement a POST method.
    - The request body must contain a username, session id, list name and titles. If any of these fields is missing an error message must be returned with a suitable code. The user already has a list present, an error with suitable code should be returned. Session id must be valid and active, else an error message should be returned. If no error is found, a list should be created with the given titles, saved in database and a success message should be returned with a suitable code.

# user/addtolist

* + - This endpoint will implement a PUT method.
    - The request body must contain a username, session id, list name and titles. If any of these fields is missing an error message must be returned with a suitable code. Session id must be valid and active, else an error message should be returned. If no error is found, a list should be updated with the given titles, saved in database and a success message should be returned with a suitable code.

# user/deletefromlist

* + - This endpoint will implement a DELETE method.
    - The request body must contain a username, session id, list name and titles. If any of these fields is missing an error message must be returned with a suitable code. Session id must be valid and active, else an error message should be returned. If no error is found, the user list should be updated by deleting the given titles from database, and a success message should be returned with a suitable code.

# user/deletelist

* + - This endpoint will implement a DELETE method.
    - The request body must contain a username, session id and list name. If any of these fields is missing an error message must be returned with a suitable code. Session id must be valid and active, else an error message should be returned. If no error is found, a list should be deleted for the user from database, and a success message should be returned with a suitable code.

# /deleteuser

* + - This endpoint will implement a DELETE method.
    - The request body must contain the admin username, session id of the admin user and username to be deleted. If any of these fields is missing an error message must be returned with a suitable code. User authorization should also be checked as Admin, if user role is not Admin then an error message should be returned with suitable code. Session id must be valid and active, else an error message should be returned. If no error is found, the user should be deleted the database, and a success message should be returned with a suitable code.

# user/logout

* + - This endpoint will implement a DELETE method.
    - The request body must contain a username and a session id.
    - If username or session id is missing from the request, return an error message with suitable code. Session id must be valid and active, else an error message should be returned.
    - If no error is found, the session id for the user must be deleted from the database and a success message must be returned.

# Database schema

# Table 1 – Username and lists

Table structure with some dummy data for example:

|  |  |
| --- | --- |
| Username (key) | List name |
| User1 | List1 |
| User2 | List2 |
| User3 | List3 |

# Table 2 – Username and session id

Table structure with some dummy data for example:

|  |  |  |
| --- | --- | --- |
| Username (key) | Session id (key) | Status |
| User1 | 12222543 | Active |
| User2 | 38628625 | Active |
| User3 | 34568683 | Active |

# Table 3 – Titles

Table structure with some dummy data for example:

|  |  |  |
| --- | --- | --- |
| List name (key) | Title (key) | Remarks (optional) |
| List1 | Friends | Cool |
| List1 | Big Bang Theory | Great |
| List1 | Game of Thrones | Favorite |
| List2 | Breaking Bad | Awesome |
| List2 | Game of Thrones | Wow |
| List3 | Friends | Long |
| List3 | Queen’s Gambit | Girl power |
| List3 | Better Call Saul | Crazy |

# Testing

* Application will be tested with Postman tool.
* All endpoints must be tested with all functionalities and for all message codes.
* It must be checked that all endpoints validate all required input fields in request and issue messages with appropriate codes.
* It must be checked that all endpoints perform CRUD operations as expected and return appropriate error codes.
* All responses must have an appropriate code as per REST principles.

# Deployment

Application will be deployed on Heroku cloud platform using free tier. Application will be available on cloud for demonstration as long as free tier is valid (~1000 hours).

# Collaboration and Maintenance

All project artifacts and assets must be stored in a git hub repository. Provisional GitHub repository is <https://github.com/abhinavjainn/QMUL_Cloud-Computing-Mini-Project>.