**API Interface**

**1: User/ User Account API**

* 1. Authenticate User Login

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/user/authenticateLogin** | | | Method=POST | Content-Type = JSON | |
|  | **Field Names** | **Data Type** | **Description** | | **Example** |
| **Input (**(Body) | Username | String | Username of user. Unique to each user | | “User\_123” |
| Password | String | Password of the user. | | “1Password” |
| **Return** | loginAuthenticationStatusObj | Object  Boolean  string  string | authenticateStatus 🡪 true if successful login  message 🡪 description of the status  jwt 🡪 json web token is issued if authentication was successful.  Payload in the jwt contains the following information.  (1) username  (2) userID  (3) role  (4) timestamp jwt was issued  (5) timestamp jwt expires | | Response: {  authenticateStatus: true,  message: 'Successful Login',  jwt: yJhbGciOiJIUzI1NiIs………  } |
| Status | Number | 200 🡪 Success, 500 🡪 Error | |  |

* 1. GetAllUsersList

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/user/getAllUsers** | | | Method=GET | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input (**(Body)) | NIL | NIL | NIL | | | NIL |
| **Return** | usersList | Collection of Objects. Each object contains user information. | User Information Objects consists of   1. UserID 🡪 UUID 2. Username 3. Password 4. Role 🡪 Either “admin” or”user” | | | Response: {  userID: “234033-234………….” username: “Admin\_User',  password: “1Password”  role: “Admin”  } |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. AddNewUser

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/user/addUser** | | | Method=POST | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input**  (Body) | newUser | Object (User) | User Object | | | {userID: “234033-234………….” username: “Admin\_User',  password: “1Password”  role: “Admin”} |
| **Return** | StatusObj | Object | Status 🡪 status of add new user | | | pass or fail |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. DeleteUser

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/user/deleteUser** | | | Method=DELETE | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input**  (URL Param) | userID | String | String 🡪 UUID | | | '00000000-0000-0000-0000-000000000000' |
| **Return** | StatusObj | Object | Status 🡪 Status of deletion, statusMsg 🡪 Description | | | statusMsg = "At Least One Admin User is needed. Unable to delete"  status: “fair”or “pass” |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. UpdateUser

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/user/updateUser** | | | Method=PUT | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input**  (URL Param) | userID | String | String 🡪 UUID | | | '00000000-0000-0000-0000-000000000000' |
| **Input**  (Body) | updatedUserDetails | Object | Object containing all user information | | | {userID: “234033-234………….” username: “Admin\_User',  password: “1Password”  role: “Admin”} |
| **Return** | StatusObj | Object | Status 🡪 Status of deletion, statusMsg 🡪 Description | | | statusMsg = "At Least One Admin User is needed. Unable to delete"  status: “fair”or “pass” |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. GetCurrentUser

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/user/getCurrentUsername** | | | Method=GET | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input** | NIL | NIL | NIL | | | NIL |
| **Return** | username | Object | Contains one property username | | | { username: “Admin\_User”} |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

**2: Audio / Audio Upload API**

* 1. GetUserAudioCollections

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/audio/getUserAudioCollections** | | | Method=GET | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input** | NIL | NIL | NIL | | | NIL |
| **Return** | collection | Collection of Objects. Each object contains user information. | User Information Objects consists of   1. filename 🡪 name of file 2. key 🡪 string info derived from upload time/date 3. audioCategory 🡪 string description of category 4. audioDescription 🡪 string description of song 5. username 🡪 username of user who uploaded file. 6. userID 🡪 user id of user who uploaded file. 7. base64 string representation of media file. | | | [{  audioCategory: "pop"  audioDescription: "best of 2000s pop …”  fileContent: “//uQZAAAAAAAA”  fileName: "Test\_1OMB\_MP3.mp3  key: "1682254380997-9"  userID: "00000000-0000-00000….."  username: "Super\_Admin"  } , {……}, {….}, ………] |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. DeleteAudioTrack

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/audio/deleteAudio** | | | Method=DELETE | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input**  (URL Param) | key | String | key 🡪 unique string representation of audio file | | | “1682254380997-9” |
| **Return** | StatusObj | Object | Status 🡪 Status of deletion | | | status: “fair”or “pass” |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. InitAudioUpload

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/audio/upload/init** | | | Method=POST | Content-Type = JSON | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input**  (Headers) | X-Content-Length | Number | depicts the size of the file to be uploaded | | |  |
| X-Content-Name | String | depicts the name of the file | | | file\_01.mp3 |
| X-Chunks-Quantity | Number | depicts the total slices/chucks the file will be boken into and sent | | | 2 |
| **Return** | fileId | String | Unique ID to represent the file to be uploaded by chunks. | | | “sdfss” |
| Status | Number | 200 🡪 Success, 500 🡪 Error | | |  |

* 1. UploadAudioChunks

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| http://ServerIP:5555**/audio/upload** | | | Method=POST | Content-Type = application/octet-stream | Authorization-Header = JWT Token | |
|  | **Field Names** | **Data Type** | **Description** | | | **Example** |
| **Input**  (Headers) | Content-Length | Number | depicts the size of the chunk to be stream | | |  |
| Content-Length | Number | Size of file | | |  |
| X-Chunk-Id | String | Used to identify the file the chunk uploaded | | | file\_01.mp3 |
| Content-Info | Object | Additional information of file uploaded, information to be saved in database together with file content. | | | username, userID, audioDescription, .. |
| **Return** | Size | Number | Size of chunk | | |  |
| Status | Number | 200 🡪 Success, | | |  |

**3. Express Server Middleware Definitions**

3.1: VerifyToken

|  |  |
| --- | --- |
| **Status** | **Action** |
| No Token Found | Return 401 status and statusMsg: "Authorization Token not send in header. Please login first" |
| Token is not a valid JWT | Return 403 and statusMsg: "Unauthorized transaction" |
| Token is valid | extract the username, userID, role and pass it to next middleware. |

3.1: ValidateAdminUserRole

Called after VerifyToken middleware is processed. Uses the extracted user role extracted by VerifyToken to determine if user is admin.

|  |  |
| --- | --- |
| **Status** | **Action** |
| User Role is Admin | Call the next middleware |
| User Role is not Admin | Return 500 and StatusMsg: <username> current role has no access rights for current action. Current role: <role>. |

**4. System Architecture**

4.1 Overview

Diagram

Description automatically generated

4.2 File System Overview

Graphical user interface

Description automatically generated

**4. Instruction to Start Application**

1. Location the docker-compose.yaml file in the root directory and run it (docker compose). This will start 4 images (frontend, backend, mongodb, mongo-express)

2. On first load, the container will initialize to two user records which can be used to log into web app

The initialize database will contain the following information

  {

        username: "Super\_Admin",

        password: "1Password",

        userID: "00000000-0000-0000-0000-000000000000",

        role: "admin",

    },

    {

        username: "User",

        password: "1Password",

        userID: "8770db80-0d4b-4edc-bf84-4076d41259c6",

        role: "user",

    },

3. The front-end application can be accessed via localhost and port 3000.

4 Changes to the database can be made using mongo express UI which is accessible via localhost:8080. For this application, the database named *audio-host* and collections named *users* and *audiofiles* are used to store data.

5. Backend API called can be made by accession [http://localhost:5555/<url](http://localhost:5555/%3curl)>

**5. Application walkthrough**

1. In the Login page, enter the username and password to login. There are no maximum attempts. (http://localhost:3000/login)

2. Upon successful login, the application will be re-directed to the accounts page. In this page, user accounts and be added, modified and deleted. These operations can only be performed by an admin user. Normal users (role=”user”) do not have the rights. (http://localhost:3000/account)

3. In the Audio Management Page, audio files uploaded by the logged in user will be displayed. Additional audio can be uploaded. Existing files can be deleted. Playback of these files is possible using the audio control player. (<http://localhost:3000/audio>)

**6. Source Codes and Project Artifacts**

The source code and artifacts can be cloned from the following GitHub repo https://github.com/mukunthan1502/Audio-Hosting-Application.git/

**7. Docker Images**

The docker images for the frontend and backed is hosted at dockerhub at the following repo mukunthan1502/mukun-repo. The mongo and mongo-express images the latest official release.