**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 | authenticateStatus 🡪 true if successful login (Boolean)  message 🡪 description of the status (string)  jwt 🡪 json web token is issued if authentication was successful. (string)  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. |

3.2: ValidateAdminUserRole

Used to verify if user is admin, handle access control requirements associated with the api request.

|  |  |
| --- | --- |
| **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**

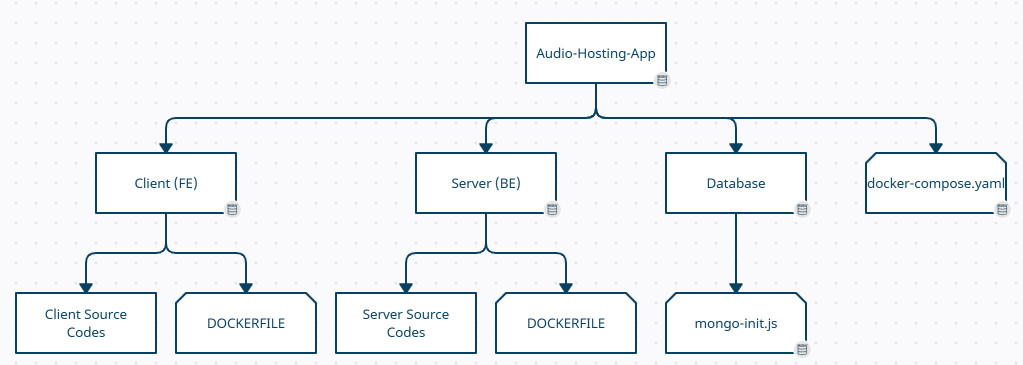
4.1 Overview

Diagram

Description automatically generated

4.2 Application File Directory Overview

The 3 directories within Audio-Hosting-App folder contain codes and artifacts relating to the frontend, backend and database. The frontend and backend folders each contain a Dockerfile for creating docker images. In the Database resides mongo-init.js script. This script will initialize the mongoDb with some initial data when the mongodb container is spawned and had no data initialize before.



4.3 Web Frontend Overview

The web frontend is developed using react framework. The frontend client runs on port 3000 and communicates with the backend service on port 5555 (configurable in docker-compose).

4.4 Nodejs Backend Overview

The backend runs on nodejs and uses express framework. The server listens for incoming requests from clients (port 5555 – configurable), processes them, and responds with the appropriate data or actions. Express provides a set of middleware functions that can be used to handle requests and responses. Middleware functions are used to perform tasks such as request access control by examining json web token(jwt) sent together with request.

The backend services also interface with a mongo DB database for persistent storage on port 27017.

4.5 Mongo DB Overview

The database is used for persistent storage in our application. The database name is “audio-host”. There are 2 collections present in the database. Communicates on port 27017. Uses latest official mongo docker image from docker hub.

|  |  |
| --- | --- |
| **Collection** | **Description** |
| users | Collection of user details information. Each document consists of username, password, userID and role. |
| audiofiles | Collection of audio details information. Each document consists of filename, key, audioCategory, audioDescription, username, userID and fileContent. |

|  |  |  |
| --- | --- | --- |
| **Document definition for user collection** | | |
| **Property** | **Data Type** | **Example** |
| username | String | “userxx” |
| password | String | “1Password” |
| userID | String (UUID) | 1234-2342… |
| role | String | “admin”or “user” |

|  |  |  |
| --- | --- | --- |
| **Document definition for audiofiles collection** | | |
| **Property** | **Data Type** | **Example** |
| filename | String | “song1.mp3” |
| key | String | “123-234324….” |
| audioCategory | String | “Rock” |
| audioDescription | String | “admin” or “user” |
| username | String | “userxx” |
| userID | String (UUID) | 1234-2342… |
| fileContent | base64 string | “//uQZAAAAAAAA” |

4.5 Mongo Express Overview

A web-based user interface (UI) tool for MongoDB that allows developers to easily manage and interact with MongoDB database using port 8080. Uses latest official mongo-express docker image from docker hub.

**5. Instruction to Start Application**

1. Run docker-compose on docker-compose.yaml.
   1. In root folder, enter cmd and type “docker compose -f docker-compose.yaml up
   2. This will start up all 4 containers.
2. Once all containers are started up. Navigate to <http://localhost:3000/login> to navigate to application.
   1. If this is the first time initializing the database, then 2 default user entries will populate the database. Initial application login will use these initial user entries. The initial user entries are defined in ./Database/mongo-init.js. Current initial user entries are as follows.

  {

        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",

    },

1. Navigate to <http://localhost:8080> to navigate to the web u interface of the mongo database. This interface allows for manual manipulation of the database. (add/remove. modify users, audio).
   1. Database used for this application is called “audio-host”
   2. Collection used are “users” and “audiofiles”

5. Backend service can be accessed using [http://localhost:5555/](http://localhost:5555/%3curl) .

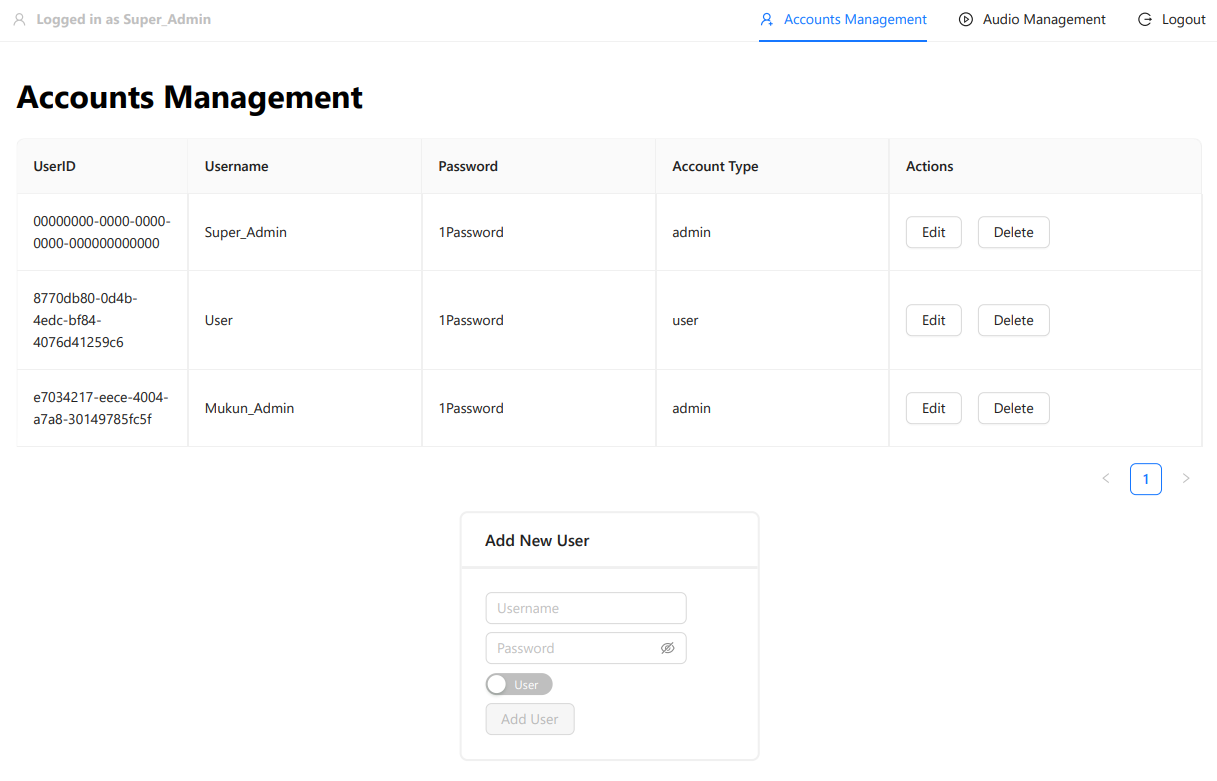
**6. Application walkthrough**

1. **Login Page (**[**http://localhost:3000/login**](http://localhost:3000/login)**)**
   1. Enter username and password to log into application.
   2. No maximum attempts penalty

Graphical user interface, application

Description automatically generated

1. **User Account Page (**[**http://localhost:3000/account**](http://localhost:3000/account)**)**
   1. Upon successful login, the application will be re-directed to the accounts page.
   2. Only admin users can add new users, edit and delete existing users.
   3. This page can also be accessed by clicking on Account Management on navigation bar.



* 1. View when click on edit.



* 1. Click on “Save” button to save changes, “Cancel” button to revert changes.

1. Audio Management Page (<http://localhost:3000/audio>)
   1. Page displays all audio files uploaded by current user only.
   2. Additional audio files can be uploaded with corresponding category and description.
      1. “Click on Upload” to upload audio file.
      2. “Add to Collection” to save file to audio collection.
   3. Click on “Delete” to remove audio file from collection.
   4. Media player on each row allows for
      1. Playback and seeking of media.
      2. Volume and playback speed adjustment.
      3. Downloading of audio file to local system.

Graphical user interface

Description automatically generated

**7. 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/>

**8. 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.