**Playlist Manager Documentation**

**Table of Contents**

**1. [Introduction]**

**2. [Dependencies]**

**3. [Configuration]**

**4. [Routes]**

**- [Home Page](#home-page)**

**- [User Registration](#user-registration)**

**- [User Login](#user-login)**

**- [User Profile](#user-profile)**

**- [Edit User Description](#edit-user-description)**

**- [Update User Photo](#update-user-photo)**

**- [Add Song](#add-song)**

**- [Get Songs](#get-songs)**

**- [Delete Song](#delete-song)**

**- [View User Profile](#view-user-profile)**

**- [User Logout](#user-logout)**

**Introduction<a name="introduction"></a>**

The Playlist Manager is a web application built using Flask, a micro web framework for Python. It allows users to register, login, manage their profiles, upload profile photos, add songs to their playlists, and view other users' profiles.

**Dependencies<a name="dependencies"></a>**

- Flask: Micro web framework for Python

- Flask-Bcrypt: Flask extension for password hashing

- Flask-Login: Flask extension for user session management

- PyJWT: JSON Web Token implementation for Python

- pymongo: Python driver for MongoDB

- Werkzeug: WSGI utility library for Python

- MongoDB: Database for storing user and song data

**Configuration<a name="configuration"></a>**

- MongoDB: The application connects to a MongoDB database hosted on MongoDB Atlas. Update the `client` variable with your MongoDB connection string.

- Secret Key: Set a unique secret key for securing the application. Update the `app.config['SECRET\_KEY']` variable.

- Upload Folder: Set the folder path for storing uploaded profile pictures. Update the `app.config['UPLOAD\_FOLDER']` variable.

**Routes<a name="routes"></a>**

**Home Page<a name="home-page"></a>**

* Route: `/`
* Description: Displays the home page with a list of all registered users.
* Method: `GET`
* Template: `index.html`

**User Registration<a name="user-registration"></a>**

* Route: `/signup`
* Description: Allows users to register by providing a username, email, password, and optional description.
* Methods: `GET`, `POST`
* Template: `signup.html`
* /signup:
* post:
* summary: User Registration
* description: Register a new user.
* requestBody:
* required: true
* content:
* application/x-www-form-urlencoded:
* schema:
* type: object
* properties:
* username:
* type: string
* email:
* type: string
* password:
* type: string
* description:
* type: string
* responses:
* '200':

description: User registered successfully.

**User Login<a name="user-login"></a>**

* Route: `/login`
* Description: Allows users to log in by providing their username and password. Generates a JWT token for authentication.

Methods:`GET`, `POST`

* Template: `login.html`
* /login:
* post:
* summary: User Login
* description: Log in with username and password to get an authentication token.
* requestBody:
* required: true
* content:
* application/x-www-form-urlencoded:
* schema:
* type: object
* properties:
* username:
* type: string
* password:
* type: string
* responses:
* '200':

description: Login successful. Returns an authentication token.

**User Profile<a name="user-profile"></a>**

* Route: `/profile/<username>`
* Description Displays the user's profile, including their description and uploaded songs.
* Method: `GET`
* Template: `profile.html`
* /profile/{username}:
* get:
* summary: Get User Profile
* description: Get the profile information for a specific user.
* parameters:
* - name: username
* in: path
* required: true
* schema:
* type: string
* responses:
* '200':

description: User profile retrieved successfully.

**Edit User Description<a name="edit-user-description"></a>**

* Route:`/editdescription`
* Description: Allows users to edit their profile description.
* Method: `POST`
* /editdescription:
* post:
* summary: Edit User Description
* description: Edit the description in the user's profile.
* requestBody:
* required: true
* content:
* application/x-www-form-urlencoded:
* schema:
* type: object
* properties:
* description:
* type: string
* responses:
* '200':

description: Description updated successfully.

**Update User Photo<a name="update-user-photo"></a>**

* Route: `/update\_photo`
* Description: Allows users to update their profile picture.
* Method: `POST`
* /update\_photo:
* post:
* summary: Update User Photo
* description: Update the user's profile picture.
* requestBody:
* required: true
* content:
* multipart/form-data:
* schema:
* type: object
* properties:
* editphotoprofile:
* type: file
* responses:
* '200':

description: Profile picture updated successfully.

**Add Song<a name="add-song"></a>**

* Route: `/addsong`
* Description Allows users to add a new song to their playlist.
* Method: `POST`
* /addsong:
* post:
* summary: Add Song
* description: Add a new song to the user's playlist.
* requestBody:
* required: true
* content:
* application/x-www-form-urlencoded:
* schema:
* type: object
* properties:
* title:
* type: string
* artist:
* type: string
* minutes:
* type: integer
* seconds:
* type: integer
* link:
* type: string
* responses:
* '200':

description: Song added successfully.

**Get Songs<a name="get-songs"></a>**

* Route: `/get\_songs`
* Description: Retrieves the list of songs associated with the logged-in user.
* Method: `GET`
* /get\_songs:
* get:
* summary: Get User Songs
* description: Get the list of songs associated with the logged-in user.
* responses:
* '200':

description: List of user's songs retrieved successfully.

**Delete Song<a name="delete-song"></a>**

* Route: `/deletesong`
* Description: Allows users to delete a song from their playlist.
* Method: `POST`
* /deletesong:
* post:
* summary: Delete Song
* description: Delete a song from the user's playlist.
* requestBody:
* required: true
* content:
* application/x-www-form-urlencoded:
* schema:
* type: object
* properties:
* songId:
* type: string
* responses:
* '200':

description: Song deleted successfully.

**View User Profile<a name="view-user-profile"></a>**

* Route: `/profile-view/<username>`
* Description: Displays the profile of another user, including their description and uploaded songs.
* Method: `GET`
* Templates:`profile-view.html`, `profile\_not\_found.html`
* /profile-view/{username}:
* get:
* summary: View User Profile (Public)
* description: View the profile and songs of another user (public view).
* parameters:
* - name: username
* in: path
* required: true
* schema:
* type: string
* responses:
* '200':

description: Public user profile retrieved successfully.

**User Logout<a name="user-logout"></a>**

* Route: `/logout`
* Description: Logs the user out by deleting the JWT token cookie.
* Method: `GET`
* /logout:
* get:
* summary: User Logout
* description: Log the user out by deleting the JWT token cookie.
* responses:
* '200':

description: Logout successful.

**Conclusion**

The Playlist Manager provides a simple and secure platform for users to manage their playlists and interact with other users. It leverages Flask and MongoDB to create a scalable and efficient web application.