**Backend Documentation**

**Introduction**

The backend of the application is responsible for handling data related to albums, artists, songs, and user information. This documentation provides details about the data models and the Flask endpoints exposed by the backend.

**Data Models**

**1. Album**

The **Album** class represents an album entity. It includes methods to convert an album object to and from a dictionary for serialization and deserialization.

**Attributes:**

* **name** (str): The name of the album.
* **imageURL** (str): URL of the album cover image.
* **release\_date** (datetime): Release date of the album.
* **total\_tracks** (int): Total number of tracks in the album.
* **songs** (List[str]): List of song names in the album.
* **artists** (List[str]): List of artist names associated with the album.

**Methods:**

* **to\_dict()**: Serializes the album object to a dictionary suitable for storing in Firestore.
* **from\_dict(source: Dict) -> Album**: Creates an **Album** instance from a dictionary.

**2. Artist**

The **Artist** class represents an artist entity. It includes methods to convert an artist object to and from a dictionary for serialization and deserialization.

**Attributes:**

* **name** (str): The name of the artist.
* **genres** (List[str]): List of genres associated with the artist.
* **image\_url** (str): URL of the artist's image.
* **popularity** (int): Popularity score of the artist.
* **albums** (List[str]): List of album names associated with the artist.

**Methods:**

* **to\_dict()**: Serializes the artist object to a dictionary suitable for storing in Firestore.
* **from\_dict(source: Dict) -> Artist**: Creates an **Artist** instance from a dictionary.

**3. Song**

The **Song** class represents a song entity. It includes methods to convert a song object to and from a dictionary for serialization and deserialization.

**Attributes:**

* **name** (str): The name of the song.
* **duration\_ms** (int): The duration of the song in milliseconds.
* **danceability** (float): Danceability score of the song.
* **energy** (float): Energy score of the song.
* **loudness** (float): Loudness level of the song.
* **tempo** (float): Tempo of the song.
* **albums** (List[str]): List of album names associated with the song.
* **artists** (List[str]): List of artist names associated with the song.

**Methods:**

* **to\_dict()**: Serializes the song object to a dictionary.
* **from\_dict(source: Dict) -> Song**: Creates a **Song** instance from a dictionary.

**4. User**

The **User** class represents a user entity. It includes methods for hashing and checking passwords, as well as converting a user object to and from a dictionary for serialization and deserialization.

**Attributes:**

* **user\_id** (Optional[str]): The ID of the user.
* **username** (str): The username of the user.
* **email** (str): The email address of the user.
* **password** (bytes): The hashed password using bcrypt.
* **settings** (Optional[Dict[str, str]]): User-specific settings.
* **created\_at** (Optional[datetime]): The timestamp when the user was created.
* **last\_login** (Optional[datetime]): The timestamp of the last login.
* **friends** (Optional[List[Dict[str, str]]]): List of friends with additional information.
* **rated\_songs** (Optional[List[str]]): List of paths to songs rated by the user.

**Methods:**

* **hash\_password(raw\_password: str) -> bytes**: Hashes the password using bcrypt.
* **check\_password(raw\_password: str) -> bool**: Checks the provided password against the hashed one in the database.
* **to\_dict() -> Dict**: Serializes the user object to a dictionary suitable for storing in Firestore.
* **from\_dict(source: Dict) -> User**: Creates a **User** instance from a dictionary.

**Flask Endpoints**

**Album Endpoints**

1. Create Album

* **Endpoint:** **/album/create**
* **Method:** **POST**
* **Request Body:**
  + **Name** (string, required): The name of the album.
  + **Image** (string, required): URL of the album cover image.
  + **Release Date** (string, required): Release date of the album (ISO 8601 format).
  + **Total Tracks** (int, required): Total number of tracks in the album.
  + **Songs** (list of strings, required): List of song names in the album.
  + **Artists** (list of strings, required): List of artist names associated with the album.
* **Response:**
  + **message** (string): Information about the operation result.
  + **album\_id** (string): ID of the newly created album.
* **Status Codes:**
  + **201 Created**: Album created successfully.
  + **400 Bad Request**: Missing or invalid request data.
  + **404 Not Found**: Could not create the album.

2. Get Album by Name

* **Endpoint:** **/album/get\_by\_name/<album\_name>**
* **Method:** **GET**
* **Path Parameters:**
  + **album\_name** (string, required): The name of the album.
* **Response:**
  + **album** (object): Album details in a dictionary format.
* **Status Codes:**
  + **200 OK**: Album retrieved successfully.
  + **404 Not Found**: Album not found.

3. Get Album by ID

* **Endpoint:** **/album/<album\_id>**
* **Method:** **GET**
* **Path Parameters:**
  + **album\_id** (string, required): The ID of the album.
* **Response:**
  + **album** (object): Album details in a dictionary format.
* **Status Codes:**
  + **200 OK**: Album retrieved successfully.
  + **404 Not Found**: Album not found.

4. Update Album

* **Endpoint:** **/album/<album\_id>**
* **Method:** **PUT**
* **Path Parameters:**
  + **album\_id** (string, required): The ID of the album to update.
* **Request Body:**
  + (Include fields to be updated, similar to the create endpoint)
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: Album updated successfully.
  + **500 Internal Server Error**: Could not update the album (server error).

5. Delete Album

* **Endpoint:** **/album/<album\_id>**
* **Method:** **DELETE**
* **Path Parameters:**
  + **album\_id** (string, required): The ID of the album to delete.
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: Album deleted successfully.
  + **500 Internal Server Error**: Could not delete the album (server error).

6. Get All Album IDs

* **Endpoint:** **/album/all**
* **Method:** **GET**
* **Response:**
  + **album\_ids** (list of strings): List of all album IDs.
* **Status Codes:**
  + **200 OK**: Album IDs retrieved successfully.

### Artist Endpoints

#### 1. Create Artist

* **Endpoint:** **/artist/create**
* **Method:** **POST**
* **Request Body:**
  + **Name** (string, required): The name of the artist.
  + **Genres** (list of strings, required): List of genres associated with the artist.
  + **Image** (string, required): URL of the artist's image.
  + **Popularity** (int, required): Popularity score of the artist.
  + **Albums** (list of strings, required): List of album names associated with the artist.
* **Response:**
  + **message** (string): Information about the operation result.
  + **artist\_id** (string): ID of the newly created artist.
* **Status Codes:**
  + **201 Created**: Artist created successfully.
  + **400 Bad Request**: Missing or invalid request data.
  + **404 Not Found**: Could not create the artist.

#### 2. Get Artist by Name

* **Endpoint:** **/artist/get\_by\_name/<artist\_name>**
* **Method:** **GET**
* **Path Parameters:**
  + **artist\_name** (string, required): The name of the artist.
* **Response:**
  + **artist** (object): Artist details in a dictionary format.
* **Status Codes:**
  + **200 OK**: Artist retrieved successfully.
  + **404 Not Found**: Artist not found.

#### 3. Get Artist by ID

* **Endpoint:** **/artist/<artist\_id>**
* **Method:** **GET**
* **Path Parameters:**
  + **artist\_id** (string, required): The ID of the artist.
* **Response:**
  + **artist** (object): Artist details in a dictionary format.
* **Status Codes:**
  + **200 OK**: Artist retrieved successfully.
  + **404 Not Found**: Artist not found.

#### 4. Update Artist

* **Endpoint:** **/artist/<artist\_id>**
* **Method:** **PUT**
* **Path Parameters:**
  + **artist\_id** (string, required): The ID of the artist to update.
* **Request Body:**
  + (Include fields to be updated, similar to the create endpoint)
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: Artist updated successfully.
  + **500 Internal Server Error**: Could not update the artist (server error).

#### 5. Delete Artist

* **Endpoint:** **/artist/<artist\_id>**
* **Method:** **DELETE**
* **Path Parameters:**
  + **artist\_id** (string, required): The ID of the artist to delete.
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: Artist deleted successfully.
  + **500 Internal Server Error**: Could not delete the artist (server error).

#### 6. Get All Artist IDs

* **Endpoint:** **/artist/all**
* **Method:** **GET**
* **Response:**
  + **artist\_ids** (list of strings): List of all artist IDs.
* **Status Codes:**
  + **200 OK**: Artist IDs retrieved successfully.

### Song Endpoints

#### 1. Create Song

* **Endpoint:** **/song/create**
* **Method:** **POST**
* **Request Body:**
  + (Include fields similar to the album and artist create endpoints, representing song details)
* **Response:**
  + **message** (string): Information about the operation result.
  + **song\_id** (string): ID of the newly created song.
* **Status Codes:**
  + **201 Created**: Song created successfully.
  + **400 Bad Request**: Missing or invalid request data.
  + **404 Not Found**: Could not create the song.

#### 2. Get Song by ID

* **Endpoint:** **/song/<song\_id>**
* **Method:** **GET**
* **Path Parameters:**
  + **song\_id** (string, required): The ID of the song.
* **Response:**
  + **song** (object): Song details in a dictionary format.
* **Status Codes:**
  + **200 OK**: Song retrieved successfully.
  + **404 Not Found**: Song not found.

#### 3. Update Song

* **Endpoint:** **/song/<song\_id>**
* **Method:** **PUT**
* **Path Parameters:**
  + **song\_id** (string, required): The ID of the song to update.
* **Request Body:**
  + (Include fields to be updated, similar to the create endpoint)
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: Song updated successfully.
  + **500 Internal Server Error**: Could not update the song (server error).

#### 4. Delete Song

* **Endpoint:** **/song/<song\_id>**
* **Method:** **DELETE**
* **Path Parameters:**
  + **song\_id** (string, required): The ID of the song to delete.
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: Song deleted successfully.
  + **500 Internal Server Error**: Could not delete the song (server error).

### User Endpoints

#### 1. Create User

* **Endpoint:** **/user/create**
* **Method:** **POST**
* **Request Body:**
  + **username** (string, required): The username of the user.
  + **email** (string, required): The email address of the user.
  + **raw\_password** (string, required): The raw password (before hashing).
  + (Include additional fields as needed, such as settings, friends, etc.)
* **Response:**
  + **message** (string): Information about the operation result.
  + **user\_id** (string): ID of the newly created user.
* **Status Codes:**
  + **201 Created**: User created successfully.
  + **400 Bad Request**: Missing or invalid request data.
  + **404 Not Found**: Could not create the user.

#### 2. Get User by ID

* **Endpoint:** **/user/<user\_id>**
* **Method:** **GET**
* **Path Parameters:**
  + **user\_id** (string, required): The ID of the user.
* **Response:**
  + **user** (object): User details in a dictionary format.
* **Status Codes:**
  + **200 OK**: User retrieved successfully.
  + **404 Not Found**: User not found.

#### 3. Update User

* **Endpoint:** **/user/<user\_id>**
* **Method:** **PUT**
* **Path Parameters:**
  + **user\_id** (string, required): The ID of the user to update.
* **Request Body:**
  + (Include fields to be updated, similar to the create endpoint)
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: User updated successfully.
  + **500 Internal Server Error**: Could not update the user (server error).

#### 4. Delete User

* **Endpoint:** **/user/<user\_id>**
* **Method:** **DELETE**
* **Path Parameters:**
  + **user\_id** (string, required): The ID of the user to delete.
* **Response:**
  + **message** (string): Information about the operation result.
* **Status Codes:**
  + **200 OK**: User deleted successfully.
  + **500 Internal Server Error**: Could not delete the user (server error).

#### 5. Get All User IDs

* **Endpoint:** **/user/all**
* **Method:** **GET**
* **Response:**
  + **user\_ids** (list of strings): List of all user IDs.
* **Status Codes:**
  + **200 OK**: User IDs retrieved successfully.
* **Response Body:**
  + **error** (string): Description of the resource not found.

**Conclusion**

This documentation provides a comprehensive overview of the backend's data models, including the **Album**, **Artist**, **Song**, and **User** classes. Additionally, it details the Flask endpoints and error handling strategies implemented in the backend. Developers can use this documentation to understand the backend structure and integrate it into the overall application.