API

I- Authentication Process

1- User Signup

URL: /auth/signup
HTTP Method: POST
This endpoint allows users to register for your application.

Body:

- **email** (string, required): The user's email address.
- **first_name** (string, required): The user's first name.
- **last_name** (string, required): The user's last name.
- **password** (string, required): The user's password.
- **profile_picture** (file, optional): The user's profile picture (image file).
- **country** (string, optional): The user's country of residence.
- **instagram** (string, optional): The user's Instagram username (without the "@" symbol).
- **facebook** (string, optional): The user's Facebook username (without the "@" symbol).
- **school** (string, optional): The user's school name.
- **tiktok** (string, optional): The user's TikTok username (without the "@" symbol).
- **level_id** (integer, required): The ID of the user's dance skill level.
- **dances** (list of integers, required): A list of IDs representing the dance styles the user practices.
- **roles** (list of integers, required): A list of IDs representing the user's roles within the application (e.g., student, instructor).

Response:

Upon successful registration, the API will return a JSON object with the following structure:

```
JSON
{
  "message": "Thanks for signing up, a verification code has been
sent to {email}",
  "access_token": "<access_token>"
```

}
message (string): A success message indicating a verification code has been sent to the user's email.

- **access_token** (string): A JSON Web Token (JWT) for authentication in subsequent requests.

Error Codes:

The API may return different error codes depending on the validation or registration failure. Specific error messages will be included in the response body.

- **400 Bad Request:**
  - Password is less than 6 characters long.
  - Missing required fields in the request body.
  - Invalid JSON format for dances or roles.
- **409 Conflict:** User with the provided email already exists.

2- User Verification

URL: /auth/verify
HTTP Method: POST
This endpoint allows users to verify their email address or reset their password, depending on the verification method used during signup.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user signup and reset password.

Body:

- **verification_code** (string, required): The verification code sent to the user's email during signup or password reset request.

Response:

The response will be a JSON object with the following structure depending on the verification outcome:

Successful Verification:

JSON

```
{

  "message": "Email verified successfully" (for verification method 'verification')

  "message": "Password reset successful, Please Login" (for method 'reset_password')

}
```

Error Codes:

- **400 Bad Request:** Verification failed or expired. (for method 'reset_password' if new_password is missing from JWT)
- **404 Not Found:** User not found with the email address associated with the JWT.
- **401 Unauthorized:** Invalid or missing JWT in the Authorization header.

3- Password Reset Request

URL: /auth/reset_password
HTTP Method: POST
“This endpoint allows users to initiate a password reset process by requesting a verification code that will be sent to their email address.”

Body:

- **email** (string, required): The user's email address associated with the account.
- **new_password** (string, required): The user's desired new password (included in the request for validation purposes, but not stored at this stage).
- **confirm_new_password** (string, required): A confirmation of the user's desired new password.

Response:

Upon successful validation and code generation, the API will return a JSON object with the following structure:

```JSON
{
  "message": "Verification code sent to your email",
  "access_token": "<access_token>"
}
```

- **message** (string): A success message indicating a verification code has been sent to the user's email.
- **access_token** (string): A JSON Web Token (JWT) for use in the subsequent verification step (/auth/verify). This JWT contains the user's email, verification code, new password (for validation during verification), expiration time, and method (reset_password).

Error Codes:

- **400 Bad Request:** Passwords do not match in the request body.
- **404 Not Found:** User with the provided email address was not found.

4- User Login

URL: /auth/login
HTTP Method: POST
This endpoint allows users to log in to your application.

Body:

- **email** (string, required): The user's email address.
- **password** (string, required): The user's password.

Response:

The response will be a JSON object with the structure depending on the login outcome:

Successful Login (Verified and Active User):

JSON

```
{
  "message": "Login successful, select your role", (or "Login
successful, please add a profile picture")
  "access_token": "<access_token>"
}
```

- **message** (string): A success message indicating login was successful.
  - The specific message may vary depending on whether the user has a profile picture or not.
- **access_token** (string): A JSON Web Token (JWT) for use in subsequent authenticated requests. This JWT includes user data like ID, names, verification status, and blocked status.

Error Codes:

- **401 Unauthorized:**
  - Invalid email or password combination.
  - User is not verified and requires verification before proceeding.
  - User account is blocked.

Response (Unverified User):

```
JSON
{
  "message": "User not verified. A verification code is sent to your
email {email}",
  "access_token": "<access_token>"
}
```

- **message** (string): A message indicating the user is not verified and a verification code has been sent.
- **access_token** (string): A JWT containing verification data (code, expiration time) to be used in the verification process (/auth/verify).

5- Get User Roles

URL: /user/roles
HTTP Method: GET

This endpoint allows authorized and verified users to retrieve their associated roles within the application.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Response:

The response will be a JSON object with the structure depending on the number of user roles:

One User Role:

JSON

```
{

  "message": "Go to role account",

  "link": "<role_slug>/<user_slug>", (link to user's role-specific account)

  "token": "<new_access_token>"

}
```

- **message** (string): A message indicating the user has one role and a link is provided to access their role-specific account.
- **link** (string): A generated link constructed by combining the user's role slug and user slug for their role-specific account. (e.g., "instructor/john-doe")
- **token** (string): A new JWT containing the user data with the retrieved role name included. This token can be used for subsequent requests within the role-specific account.

Multiple User Roles:

JSON

```
{"roles": [

    {"role_id": 1, "role_name": "Student"},
    {"role_id": 2, "role_name": "Instructor"}
  ]}
```

- **roles** (list of objects): A list of objects containing information for each user role:
  - **role_id** (integer): The ID of the user role.
  - **role_name** (string): The name of the user role.

Error Codes:

- **401 Unauthorized:**
  - Invalid or missing JWT in the Authorization header.
  - User is not verified.
- **404 Not Found:**
  - User not found with the ID extracted from the JWT.
  - No roles are associated with the user.

6- Select User Role

URL: /user/select-role
HTTP Method: POST
This endpoint allows users with multiple roles to select their preferred role and obtain a new JWT containing the selected role information.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Body:

- **role_id** (integer, required): The ID of the role the user wants to select.

Response:

The response will be a JSON object with the structure:

```
JSON
{
  "message": "Go to role account",
  "link": "<role_slug>/<user_slug>", (link to user's role-specific
account)
  "token": "<new_access_token>"
}
```

- **message** (string): A message indicating the user selection was successful and a link is provided to access their role-specific account.
- **link** (string): A generated link constructed by combining the user's selected role slug and user slug for their role-specific account. (e.g., "instructor/john-doe")
- **token** (string): A new JWT containing the user data with the selected role name included. This token can be used for subsequent requests within the role-specific account.

Error Codes:

- **401 Unauthorized:**
  - Invalid or missing JWT in the Authorization header.
  - User is not verified.
- **404 Not Found:**
  - User not found with the ID extracted from the JWT.
  - No roles are associated with the user.
  - Selected role ID is not found among the user's roles.

II - User Process (Competitor)

1- User Profile

a- Get User Profile

URL: /<role>/<slug>`
HTTP Method: GET
This endpoint allows users to retrieve the public profile information of another user based on their role and user slug (unique identifier).

Path Parameters:

- **role** (string, required): The user's role (e.g., "student", "instructor").
- **slug** (string, required): The user's unique slug identifier.

Response:

Upon successful retrieval, the API will return a JSON object containing the user's profile data:

```
JSON
{
  "user_profile": {
    "user_firstname": "<first name>",
    "user_lastname": "<last name>",
    "user_email": "<email address>",
    "user_country": "<country>", (optional)
    "user_instagram": "<instagram username>", (optional)
    "user_facebook": "<facebook username>", (optional)
    "user_tiktok": "<tiktok username>", (optional)
    "user_school": "<school name>", (optional)
    "user_profile_picture": "<path to profile picture>", (optional)
    "level": "<user's dance skill level name>", (optional)
    "user_dances": ["Dance 1", "Dance 2", ...], (list of user's
dance styles)
    "user_roles": ["Role 1", "Role 2", ...], (list of user's roles)
    "first_place": <number of first place achievements>, (integer)
    "second_place": <number of second place achievements>, (integer)
    "third_place": <number of third place achievements>, (integer)
    "current_points": <user's current league points>, (integer)
    "evolution_points": <difference between current and previous
points>, (integer)
  }
}
```

Error Codes:

- **404 Not Found:**
  - User with the provided slug was not found.
  - User was found but does not have the specified role (case-insensitive matching).

b- Update User Information

URL: /upload_user_information
HTTP Method: PUT
This endpoint allows authorized and verified users to update their profile information.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Body:

- **first_name** (string): The user's first name.
- **last_name** (string): The user's last name.
- **country** (string): The user's country of residence.
- **level_id** (integer): The ID of the user's dance skill level (refer to dance level data model).
- **instagram** (string): The username for the user's Instagram profile.
- **facebook** (string): The username for the user's Facebook profile.
- **tiktok** (string): The username for the user's TikTok profile.
- **dances** (string, JSON list): An array of dance IDs representing the user's dance styles (refer to dance data model).

Response:

Upon successful update, the API will return a JSON object with the following structure:

```
JSON
{
  "message": "User information updated successfully",
  "link": "/<new_user_slug>"  (link to the user's updated profile)
}
```

- **message** (string): A confirmation message indicating the user information was successfully updated.
- **link** (string): A link to the user's profile page, reflecting any updates to the user's slug (unique identifier).

Error Code:

- **404 Not Found:** User not found with the ID extracted from the JWT.

Logic and Security Considerations:

- The implementation prioritizes updating only the fields with provided values in the request body, to avoid accidentally overwriting existing data in case a checkbox is left unchecked on the front-end.
- However, commented-out code snippets show an alternative approach where each field is updated only if a value is provided. This approach might be preferable depending on your specific requirements.
- If the user's first name or last name is updated, a new unique slug is generated to maintain uniqueness within user profile URLs.

c- Upload Profile Picture

URL: /upload_profile_picture
HTTP Methods: POST, PUT
This endpoint allows authorized and verified users to upload a profile picture.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Body:

- **profile_picture** (file, required): The user's profile picture image file. Supported image formats likely depend on your backend configuration (e.g., JPEG, PNG).

Response:

Upon successful upload, the API will return a JSON object with the following structure:

```JSON
{
  "message": "Profile picture updated successfully",
  "file_path": "<path to the uploaded profile picture>"
}
```

- **message** (string): A confirmation message indicating the profile picture was successfully uploaded.
- **file_path** (string): The path (relative or absolute depending on your implementation) where the uploaded profile picture is stored.

Error Codes:

- **400 Bad Request:** No profile picture was included in the request.
- **404 Not Found:** User not found with the ID extracted from the JWT.

2- Pictures

a- Get User Pictures

URL: /user/<string:slug>/pictures
HTTP Method: GET
This endpoint allows retrieving all the pictures associated with a user's profile.

Path Parameter:

- **slug** (string, required): The user's unique slug identifier.

Response:

Upon successful retrieval, the API will return a JSON object with the following structure:

```JSON
{
  "user_pictures": [
    {"id": 1, "picture_path": "<path/to/picture1.jpg"},
    {"id": 2, "picture_path": "<path/to/picture2.jpg"},
    ...
  ]
}
```

- **user_pictures** (list): A list of objects containing information about the user's pictures:
  - **id** (integer): The ID of the picture in the database.
  - **picture_path** (string): The path (relative or absolute depending on your implementation) where the picture is stored.

Error Code:

- **404 Not Found:** User with the provided slug was not found.

Important Note:

- This endpoint retrieves potentially many pictures. Be mindful of performance implications on the server-side and potential bandwidth usage on the client-side, especially for mobile clients.
- Consider implementing pagination mechanisms to retrieve pictures in smaller chunks if you expect a large number of pictures per user.

b- Upload User Picture

URL: /upload_user_picture
HTTP Method: POST
“This endpoint allows authorized and verified users to upload a picture associated with their profile but not specifically designated as their profile picture.”

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login

Body:

The request body should be a multipart form data encoded payload containing a single file field:

- **picture** (file, required): The user's picture to upload. Supported image formats likely depend on your backend configuration (e.g., JPEG, PNG).

Response:

Upon successful upload, the API will return a JSON object with the following structure:

```
JSON
{
  "message": "Picture uploaded successfully",
  "filename": "<filename of the uploaded picture>"
}
```

- **message** (string): A confirmation message indicating the picture was successfully uploaded.
- **filename** (string): The filename of the uploaded picture.

Error Codes:

- **400 Bad Request:**
  - No picture was included in the request ('picture' not in request.files).
  - An empty file was selected (picture_file.filename == '').

c- Delete User Picture

URL: /delete_user_picture/<string:filename>
HTTP Method: DELETE
This endpoint allows authorized and verified users to delete pictures associated with their profile.

Path Parameter:

- **filename** (string, required): The filename of the picture to delete.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Response:

Upon successful deletion:

- **200 OK:** A JSON object with a message indicating success.

Error Codes:

- **400 Bad Request:** The request might be invalid due to malformed data. This is not implemented in the provided code but could be added for robustness.
- **404 Not Found:**
    - The picture filename was not found in the user's associated storage folder.
    - The picture record was not found in the database.
- **500 Internal Server Error:**
    - An error occurred while trying to delete the picture file from storage.
    - An error occurred while trying to delete the picture record from the database.

3- Videos

a- Get User Videos

URL: /user/<string:slug>/videos
HTTP Method: GET
This endpoint allows retrieving all the videos associated with a user's profile.

Path Parameter:

- **slug** (string, required): The user's unique slug identifier.

Response:

Upon successful retrieval, the API will return a JSON object with the structure:

```
JSON
{
  "user_videos": [
    {"id": 1, "video_path": "<path/to/video1.mp4"},
    {"id": 2, "video_path": "<path/to/video2.mp4"},
    ...
  ]
}
```

- **user_videos** (list): A list of objects containing information about the user's videos:
  - **id** (integer): The ID of the video in the database.
  - **video_path** (string): The path (relative or absolute depending on your implementation) where the video is stored.

Error Code:

- **404 Not Found:** User with the provided slug was not found.

b- Upload User Video

URL: /upload_user_video
HTTP Method: POST
This endpoint allows authorized and verified users to upload a video associated with their profile.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Body:

The request body should be a multipart form data encoded payload containing a single file field:

- **video** (file, required): The user's video to upload. Supported video formats likely depend on your backend configuration (e.g., MP4). Consider security implications of allowing arbitrary file uploads (e.g., potential for malicious file types).

Response:

Upon successful upload, the API will return a JSON object with the following structure:

JSON

```
{
  "message": "Video uploaded successfully",
  "filename": "<filename of the uploaded video>"
}
```

- **message** (string): A confirmation message indicating the video was successfully uploaded.
- **filename** (string): The filename of the uploaded video.

Error Codes:

- **400 Bad Request:**
  - No video was included in the request ('video' not in request.files).
  - An empty file was selected (video_file.filename == '').

c- Delete User Video

URL: /delete_user_video/<string:filename>
HTTP Method: DELETE
This endpoint allows authorized and verified users to delete videos associated with their profile.

Path Parameter:

- **filename** (string, required): The filename of the video to delete.

Authentication:

This endpoint requires a valid JWT (JSON Web Token) in the Authorization header. You can obtain a JWT upon successful user login.

Response:

Upon successful deletion:

- **200 OK:** A JSON object with a message indicating success.

Error Codes:

- **400 Bad Request:** The request might be invalid due to malformed data. This is not implemented in the provided code but could be added for robustness.
- **404 Not Found:**
  - The video filename was not found in the user's associated storage folder.
  - The video record was not found in the database.
- **500 Internal Server Error:**
  - An error occurred while trying to delete the video file from storage.
  - An error occurred while trying to delete the video record from the database.

4- Teams

a- Get user Team

b- Create Competitor Team

c- Delete User Team