API Documentation

Login Register APIs

/register - User Registration

Description: This API is used to register a new user into the system.

Method: POST

URL : **http://127.0.0.1:5000/register**

Body (JSON):

```
{
  "name": "John Doe",
  "email": "johndoe@gmail.com",
  "password": "password123"
}
```

- **Inputs**:
  - name (string): The user's name.
  - email (string): The user's email. Must end with @gmail.com.
  - password (string): The user's password. Must be at least 8 characters long.
- **Processes**:
  - Validates the inputs.
  - Hashes the password using bcrypt.
  - Generates a random verification code (OTP).
  - Registers the user into the database, storing the verification code and its expiry date.
  - Sends the verification code to the user's email.
- **Outputs**:
  - On success: {'message': 'User registered successfully! Verification code sent to email.'}
  - On error (e.g., email already exists or database connection error): {'error': 'Error message'}

/verify-code - Code Verification

Description: This API is used to verify the OTP sent to the user's email during registration.

Method: POST

URL : http://127.0.0.1:5000/verify-code

Body (JSON):

```
{
  "email": "johndoe@gmail.com",
  "code": "123456"
}
```

- **Inputs**:
  - email (string): The user's email.
  - code (string): The verification code sent to the user's email.
- **Processes**:
  - Validates the email and code.
  - Checks if the code has expired.
  - If the code is valid, updates the email verification status in the database.
- **Outputs**:
  - On success: {'message': 'Email verified successfully!'}
  - On error (e.g., invalid or expired code): {'error': 'Error message'}

/resend-verification-code - Resend Verification Code

Description: This API is used to resend the OTP verification code to the user's email in case the code was not received or has expired.

Method: POST

URL : http://127.0.0.1:5000/resend-verification-code

Body (JSON):

```
{
  "email": johndoe@gmail.com
}
```

- **Inputs**:
  - email (string): The user's email requesting the verification code.
- **Processes**:
  - Validates the email.
  - If the email exists and is not verified, generates a new verification code and sends it via email.
  - Updates the verification code and expiry date in the database.
- **Outputs**:
  - On success: {'message': 'Verification code resent successfully.'}
  - On error (e.g., email not found or already verified): {'error': 'Error message'}

/login - User Login

Description: This API allows registered users to log in to the system.

Method: POST

URL : http://127.0.0.1:5000/login

Body (JSON):

```
{
  "email": "johndoe@gmail.com",
  "password": "password123"
}
```

- Inputs:
  - email (string): The user's email.
  - password (string): The user's password.
- Processes:
  - Validates the email and password.
  - Checks if the email is verified.
  - If the password is correct and the email is verified, logs in the user by storing user_id in the session.
- Outputs:
  - On success: {'message': 'Login successful!'}

- On error (e.g., incorrect password or unverified email): {'error': 'Error message'}

send_verification_email Function

Description: This function sends an email containing the OTP verification code to the user's email address.

/forgot-password - Forgot Password Request

Description: This API is used to request a password reset by sending a verification code to the user's email.

- **Method**: POST
- **URL**: http://127.0.0.1:5000/forgot-password

Body (JSON):

```
{

  "email": "johndoe@gmail.com"

}
```

Inputs:

- **email (string)**: The user's registered email. This is required to find the user in the system.

Processes:

- Checks if the email exists in the database.
- Generates a new verification code and sets an expiry time (10 minutes).
- Sends the verification code to the user's email.

Outputs:

- On success: `{'message': 'Password reset code sent to email.'}`

/verify-reset-code - Verify Password Reset Code

Description: This API is used to verify the reset code sent to the user's email.

- **Method**: POST
- **URL**: http://127.0.0.1:5000/verify-reset-code

Body (JSON):

```
{
    "email": "johndoe@gmail.com",
    "reset_code": "123456"
}
```

Inputs:

- **email (string)**: The user's registered email.
- **reset_code (string)**: The 6-digit verification code sent to the user's email.

Processes:

- Checks if the reset code matches the one stored in the database for the given email.
- Verifies that the reset code has not expired.

Outputs:

- On success: `{'message': 'Reset code verified successfully!'}`
- On error (e.g., invalid or expired reset code): `{'error': 'Error message'}`

/reset-password - Reset Password

Description: This API is used to reset the user's password after successfully verifying the reset code.

- **Method**: POST
- **URL**: http://127.0.0.1:5000/reset-password

Body (JSON):

```
{
    "email": "johndoe@gmail.com",
    "new_password": "newpassword123"
}
```

Inputs:

- **email (string)**: The user's registered email.
- **new_password (string)**: The new password for the user. Must be at least 8 characters long.

Processes:

- Verifies that the email is correct and checks if the password is valid.
- Hashes the new password using bcrypt.
- Updates the user's password in the database and clears the verification code fields.

Outputs:

- On success: `{'message': 'Password reset successfully!'}`
- On error (e.g., invalid email or database error): `{'error': 'Error message'}`

/auth/google - Google Login

Description: This API redirects users to Google's OAuth 2.0 login page, allowing them to sign in using their Google account.

Method: GET
URL: http://127.0.0.1:5000/auth/google

- **Inputs**: None
- **Processes**:
    - Redirects the user to the Google OAuth 2.0 authorization page.
    - The user grants permission to access their Google profile and email.
- **Outputs**:
    - Redirects the user to Google's OAuth 2.0 login page.

/auth/google/callback - Google OAuth Callback

Description: This API handles the callback from Google OAuth after a user signs in. It exchanges the authorization code for an access token, retrieves the user's profile, and stores or verifies the user in the database.

Method: GET
URL: http://127.0.0.1:5000/auth/google/callback

- **Inputs**:
  - Query parameter `code`: The authorization code returned by Google after successful login.
- **Processes**:
  - Exchanges the authorization code for an access token and ID token.
  - Verifies the ID token and retrieves the user's profile information (email, name).
  - Checks if the user is already registered in the database. If not, creates a new user with a randomly generated password.
  - Initiates a session for the logged-in user.
- **Outputs**:
  - On success: `{"message": "Login successful!", "user": {"email": "<email>", "name": "<name>"}}`
  - On error: `{"error": "Error message"}` (e.g., failed token exchange, invalid issuer, or database connection failure)

/logout - User Logout

Description: This API is used to log out the currently logged-in user by removing their session.
Method: POST
URL: http://127.0.0.1:5000/logout

- **Inputs:**
  - No input required in the body. The session is used to identify the logged-in user.
- **Processes:**
  - Checks if a user is currently logged in by verifying the presence of `user_id` in the session.
  - If found, it removes the `user_id` from the session to log the user out.
- **Outputs:**

- On success:

```
{ "message": "Logout successful." }
```

- On error (user not logged in):

```
{ "error": "User not logged in." }
```

Admin APIs

/users - Get All Users

Description:
This API retrieves a list of all users along with their image upload counts. Only accessible to admin users.

Method: GET
URL: http://127.0.0.1:5000/users

Inputs:

- Requires a valid session with an admin user logged in.

Processes:

- Checks if the requesting user is logged in and has an admin role.
- Queries the database for all users and counts the number of images each user has uploaded.

Outputs:

- On success:

```
"users": [
  {
    "user_id": 1,
    "name": "John Doe",
    "email": "john@example.com",
    "image_count": 5
```

```
      },
      ...
    ]
  }
```

/Admins - Get All Admins

Description:
This API retrieves a list of all users who have an admin role. Only accessible to admin users.

Method: GET
URL: http://127.0.0.1:5000/Admins

Inputs:

- Requires a valid session with an admin user logged in.

Processes:

- Validates the session and role.
- Fetches all users with the role `'Admin'`.

Outputs:

- On success:

```
{
  "Admins": [
    {
      "name": "Admin Name",
      "email": "admin@example.com"
    },
    ...
  ]
}
```

- On failure: Appropriate error messages with status codes (`401`, `403`, `500`).

/admin/promote-to-admin - Promote a User to Admin

Description:
This API promotes an existing user to admin by updating their role. Only accessible to admin users.

Method: POST
URL: http://127.0.0.1:5000/admin/promote-to-admin

Body (JSON):

```
{
  "email": "user@example.com"
}
```

Inputs:

- `email` (string): The email address of the user to be promoted.
- Requires a valid session with an admin user logged in.

Processes:

- Validates admin session.
- Checks if the user exists and is not already an admin.
- Updates the user role to `'Admin'`.

Outputs:

- On success:

```
{ "message": "User promoted to admin successfully!" }
```

/admin/demote-to-user - Demote an Admin to Regular User

Description:
This API allows an admin user to demote an existing admin to a regular user by updating their role. It is only accessible by admin users.

- **Method:** POST

- **URL:** http://127.0.0.1:5000/admin/demote-to-user
- **Body (JSON):**

```
{
  "email": "user@example.com"
}
```

- **Inputs:**
  - `email` (string): The email address of the user to be demoted.
- **Process:**
  - Validates if the current user is an admin.
  - Checks if the user exists and if the user is currently an admin.
  - Updates the user role to `User`.
- **Outputs:**
  - **On success:**

```
{
  "message": "Admin demoted to regular user successfully!"
}
```

  - **On failure:**
    - Returns an error if the user does not exist or is not an admin, or if the current user is not an admin.

`/system-stats` - Get System Stats

Description:
This API retrieves the system statistics, including the total number of users, generated images, and detected images in the system. This API is only accessible to admin users.

- **Method:** `GET`
- **URL:** http://127.0.0.1:5000/system-stats
- **Inputs:**
  - Requires a valid session with an admin user logged in.
- **Process:**
  - Validates if the current user is an admin.
  - Retrieves the total number of users, generated images, and detected images from the database.
- **Outputs:**
  - **On success:**

```
{
  "total_users": 100,
  "total_generated_images": 500,
  "total_detected_images": 300
}
```

- **On failure:**
  - Returns an error if the user is not logged in or if the user is not an admin.

`/delete-user-by-email`-Delete a User by Email

Description:
This API allows an admin to delete a user by their email address, removing their data from the system, including images generated and detected by the user. It also deletes the user's corresponding folders. This API is only accessible by admin users.

- **Method:** `DELETE`
- **URL:** `http://127.0.0.1:5000/delete-user-by-email`
- **Body (JSON):**

```
{
  "email": "user@example.com"
}
```

- **Inputs:**
  - `email` (string): The email address of the user to be deleted.
- **Process:**
  - Validates if the current user is an admin.
  - Checks if the user exists and retrieves their user ID.
  - Deletes the images related to the user from the `Images_generation` and `Images_detection` tables.
  - Deletes the user from the `[User]` table.
  - Deletes the corresponding folders (`generate` and `detect`) for the user.
- **Outputs:**
  - **On success:**

```
{
  "message": "User and their images have been deleted successfully!"
```

```
}
```

- **On failure:**
  - Returns an error if the user does not exist, if the user is an admin and tries to delete their own account, or if the current user is not an admin.

Detection APIs

`/upload-image-local` – Upload a Local Image and Detect Content

- **Description**:
  Enables a logged-in user to upload an image from their device. The image is processed, detected, saved, and recorded in the database.
- **Method**: `POST`
- **URL**: `http://127.0.0.1:5000/upload-image-local`
- **Authentication Required**:
- **Request Body (form-data)**:

```
Key: image  |  Value: (uploaded image file)
```

- **Process**:
  - Accepts a local image file upload.
  - Converts to RGB if needed.
  - Detects the image using the model.
  - Saves the image in the user's detection folder.
  - Records the result in the `Images_detection` table.

- **Successful Response**:

```
{
  "message": "Image uploaded and detected successfully!",
  "predicted_class": "Dog",
  "confidence": "94.62%"
}
```

◇ `/upload-URL-image` – Upload an Image from a URL and Detect Content

Description:
Allows a logged-in user to upload an image using a direct URL. The image is analyzed by a detection model, saved to disk, and the detection result is stored in the database.

Method: `POST`
URL: `http://127.0.0.1:5000/upload-URL-image`
Authentication Required: (User must be logged in)

Request Body (JSON):

```
{
  "image_url": "https://example.com/image.jpg"
}
```

Process:

- Validates that the user is logged in via session.
- Downloads the image from the provided URL.
- Converts it to RGB if needed.
- Preprocesses the image and runs it through a pre-trained detection model.
- Saves the image in a folder named after the user (`user_<id>`).
- Stores the image path and detection result in the `Images_detection` table.

Successful Response (201 Created):

```
{
  "message": "Image uploaded and detected successfully!",
  "predicted_class": "Cat",
  "confidence": "97.83%"
}
```

Error Responses:

- `401 Unauthorized` – User is not logged in

```
{
  "error": "Please log in first."
}
```

- `400 Bad Request` – Missing or invalid image URL

```
{
  "error": "Please provide an image URL."
}
```

- `500 Internal Server Error` – Any other processing error

```
{
  "error": "Error processing image: <error_message>"
}
```

- `400 Bad Request` – Failed to download image

```
{
  "error": "Failed to download image: <error_message>"
}
```

`/user-detection-images` – Get User's Detected Images (Base64)

- **Description**:
  Retrieves all detected images for the currently logged-in user, along with their detection results. Images are returned in Base64 format.
- **Method**: `GET`

- **URL**: http://127.0.0.1:5000/user-detection-images
- **Authentication Required**:
- **Process**:
    - Fetches all image paths and detection results from `Images_detection` table for the user.
    - Converts each image file to Base64 if the file exists.
    - Returns a list of detected images and labels.
- **Successful Response**:


```
  {
    “image_id”: 5
    "detection_result": "Dog",
    "image": "..."
  },
  {
    “image_id”: 2
    "detection_result": "Car",
    "image": "..."
  }
]
```


/delete-detection-image - Delete Detection Image by ID

Description: Deletes a specific detection image belonging to the currently logged-in user by image ID.

Method: DELETE
URL: http://127.0.0.1:5000/delete-detection-image

Body (JSON):


```
  "image_id": 5
}
```


Inputs:

- `image_id (int): The ID of the image to delete.`
- `Requires a valid session with user_id.`

Processes:

- Verifies that the image belongs to the current user.
- Deletes the image from disk if it exists.
- Deletes the image record from the Images_detection table.

Outputs:

- On success:

```
{ "message": "Image deleted successfully" }
```

- On error:

```
{ "error": "Image ID is required" }
```

or

```
{ "error": "Image not found or does not belong to user" }
```

or

```
{ "error": "Database connection failed" }
```

/delete-all-detection-images - Delete All Detection Images

Description: Deletes all detection images for the currently logged-in user.

Method: DELETE
URL: http://127.0.0.1:5000/delete-all-detection-images

Body:

- No body required.

Inputs:

- Requires a valid session with user_id.

Processes:

- Fetches all image paths for the current user.
- Deletes all corresponding image files from disk.
- Deletes all records from the Images_detection table for the user.

Outputs:

- On success:

```
{ "message": "All detection images deleted successfully" }
```

- On error:
- { "error": "Unauthorized" }

or

```
{ "error": "Database connection failed" }
```

Generation APIs

/generate-image - Generate AI Image from Description

Description: Generates an image using a text prompt (description) via a diffusion model (like Stable Diffusion), and saves the generated image for the logged-in user.
Method: POST
URL: http://127.0.0.1:5000/generate-image
Body (JSON):

```
{
  "description": "A futuristic city at night"
}
```

- **Inputs:**
  - `description` (string): A prompt used for generating the image.
- **Processes:**
  - Verifies user session.
  - Passes the prompt to the AI image generation pipeline.
  - Saves the generated image in a specific folder for the user.
  - Stores the image path and description in the database.
- **Outputs:**
  - On success: Returns the generated image as a PNG file via `send_file`.
  - On error: `{"error": "Error message"}`

`/send-last-generated-image` – Send the Last Generated Image to User's Email

Description:
Allows a logged-in user to retrieve the most recently generated image and sends it to their registered email address.

Method: `POST`
URL: `http://127.0.0.1:5000/send-last-generated-image`
Authentication Required: (User must be logged in)

Request Body:
None required (empty or `{}` is accepted)

Process:

- Validates that the user is authenticated via session.
- Connects to the database.
- Queries the `Images_generation` table for the last generated image by the user (based on `image_id DESC`).
- Retrieves the user's email from the `User` table.
- Sends the image and its description to the user's email using `send_image_email()`.

Successful Response (200 OK):

```
{
  "message": "Image sent to email successfully."
}
```

Error Responses:

- `401 Unauthorized` – User is not logged in

```
{
  "error": "Unauthorized"
}
```

- `404 Not Found` – No image found for user

```
{
  "error": "No generated image found"
}
```

- `404 Not Found` – User email not found

```
{
  "error": "User email not found"
}
```

- `500 Internal Server Error` – Any unexpected issue

```
{
  "error": "Database connection failed"
}
 or
{
  "error": "<detailed_error_message>"
}
```

`/user-generated-images` - Get All User Generation Images

Description: This API retrieves all generation images that belong to the currently logged-in user in base64 format.

Method: `GET`
URL: `http://127.0.0.1:5000/user-generated-images`

Body:

- No body required.

Inputs:

- Requires a valid session with `user_id`.

Processes:

- Connects to the database.
- Selects all image records from the `Images_generation` table for the logged-in user.
- Reads image files from disk.
- Encodes each image into base64 format.
- Returns image list including image ID, description, and base64-encoded image.

Outputs:

- On success: Returns a list of images in the following format:

```
[
  {
    "image_id": 1,
    "description": "Example image",
    "image": "..."
  },
  {
    "image_id": 2,
    "description": "Another one",
    "image": "..."
  }
]
```

- On error (e.g. unauthorized or database issues):

```
{ "error": "Unauthorized" }
```

or

```
{ "error": "Database connection failed" }
```

/delete-generation-image - Delete Generation Image by ID

Description: Deletes a specific generation image belonging to the currently logged-in user by image ID.

Method: `DELETE`
URL: http://127.0.0.1:5000/delete-generation-image

Body (JSON):

```
{
  "image_id": 7
}
```

Inputs:

- `image_id` (int): The ID of the generation image to delete.
- Requires a valid session with `user_id`.

Processes:

- Verifies that the image belongs to the current user.
- Deletes the image from disk.
- Deletes the image record from the `Images_generation` table.

Outputs:

- On success:

```
{ "message": "Image deleted successfully" }
```

- On error:

```
{ "error": "Image ID is required" }
```

or

```
{ "error": "Image not found or does not belong to user" }
```

or

```
{ "error": "Database connection failed" }
```

`/delete-all-generation-images` - Delete All Generation Images

Description: Deletes all generation images for the currently logged-in user.

Method: `DELETE`
URL: `http://127.0.0.1:5000/delete-all-generation-images`

Body:

- No body required.

Inputs:

- Requires a valid session with `user_id`.

Processes:

- Fetches all image paths for the current user.
- Deletes all corresponding image files from disk.
- Deletes all records from the `Images_generation` table for the user.

Outputs:

- On success:

```
{ "message": "All generation images deleted successfully" }
```

- On error:

```
{ "error": "Unauthorized" }
```

or

```
{ "error": "Database connection failed" }
```