**Documentation for Flask Application**

**Table of Contents**

1. Dependencies
2. Application Configuration
3. Azure Blob Storage Configuration
4. Database Configuration and Connection Pooling
5. Token Required Decorator
6. Endpoints
   * User Signup (/user/signup)
   * User Signin (/user/signin)
   * Image Upload (/upload\_image)
   * Forgot Password (/user/forgot\_password)
   * Reset Password (/user/reset\_password/<token>)
   * Update Password (/user/update\_password)
   * Change Password (/user/change\_password)

**1. Dependencies**

* **Flask:** Flask is a lightweight web framework for Python. It provides tools, libraries, and patterns to create web applications. In this code, Flask is used to define routes, handle HTTP requests, and manage the application.
* **Werkzeug:** Werkzeug is a utility library for WSGI (Web Server Gateway Interface) applications in Python. In this code, Werkzeug is used for security functions (generate\_password\_hash and check\_password\_hash) and for handling file uploads (secure\_filename).
* **Azure Storage Blob:** Azure Storage Blob is a library for interacting with Azure Blob Storage. It is used to upload images to Azure Blob Storage in the upload\_to\_azure\_blob function.
* **MySQL Connector:** MySQL Connector is a MySQL database driver for Python. It is used to connect to a MySQL database, execute queries, and manage database connections in this code.
* **JWT (JSON Web Tokens):** JSON Web Tokens (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. It is used for user authentication and authorization in this code. The jwt library is used to encode and decode JWTs.
* **Secrets:** The secrets module is used to generate cryptographically strong random numbers suitable for managing secrets such as authentication tokens. In this code, it is used to generate a random secret key for securing JWTs.
* **OS:** The os module provides a way of using operating system-dependent functionality. In this code, it is used to access environment variables or perform OS-level operations.
* **Functools:** The functools module provides higher-order functions and operations on callable objects. In this code, the wraps decorator from functools is used to preserve the original function's metadata when creating a decorator.

**2. Application Configuration**

* A Flask application is created, and a secret key is generated for securing sessions.
* The Flask application is configured with a secret key for JWT token encoding and decoding.

**3. Azure Blob Storage Configuration**

* Azure Storage connection string and container name are defined for uploading images to Azure Blob Storage.

**4. Database Configuration and Connection Pooling**

* MySQL database connection parameters (host, user, password, database) are configured.
* A connection pool is created using MySQL Connector's pooling feature.

**5. Token Required Decorator**

* The token\_required decorator checks for the presence of a valid JWT token in the request headers.
* If a valid token is present, the decorator decodes the token and passes the current user to the decorated function.
* Used to secure the /protected route.

**6. Endpoints**

* **User Signup (/user/signup)**
  + **Method:** POST
  + Expects a JSON payload.
  + Creates a new user in the database with hashed password.
  + Requires *application\_id* and *client\_id* headers for successful user creation.

**Request Body:**

fullname (string): Full name of the user.

username (string): Unique username for the user.

password (string): User's password.

email (string): User's email address.

phone\_number (integer): User's phone number.

**Response:**

201 Created: User created successfully.

*{*

*"message": "User created successfully"*

*}*

400 Bad Request: Missing fields.

*{*

*"message": "Missing fields",*

*"missing": ["field1", "field2"]*

*}*

500 Internal Server Error: Failed to create user.

*{*

*"message": "Failed to create user"*

*}*

* **User Signin (/user/signin)**
  + **Method:** POST
  + Expects a JSON payload with *username* and *password.*
  + Validates user credentials against the database and returns a JWT token for successful authentication.

**Request Body:**

username (string): User's username.

password (string): User's password.

**Response:**

200 OK: Login successful.

*{*

*"message": "Login successful",*

*"token": "jwt\_token\_here"*

*}*

401 Unauthorized: Invalid username or password.

*{*

*"message": "Invalid username or password"*

*}*

500 Internal Server Error: Database error.

*{*

*"message": "Database error",*

*"error": "error\_details\_here"*

*}*

* **Image Upload (/upload\_image)**
  + **Method:** POST
  + Expects a file with the key image in the request.
  + Uploads the image to Azure Blob Storage and returns the image URL.

**Request Body:**

**Form-data:**

Key: file name.

Value: user selected file.

**Response:**

200 OK: Image uploaded successfully and image uploaded path displayed.

*{*

*"message": "Image uploaded successfully",*

*"url": "https://azure\_blob\_storage\_url/image\_filename"*

*}*

400 Bad Request: either image part or selected file is missing.

*{*

*"message": "No image part" or "No selected file"*

*}*

401 Unauthorized: Invalid or no token.

*{*

*"message": "Token is missing or invalid"*

*}*

500 Internal Server Error: Database error.

*{*

*"message": "Error details here"*

*}*

* **Forgot Password (/user/forgot\_password)**
  + **Method:** POST
  + Expects registered email in the request.
  + Sends a password reset link to the user's email.

**Request Body:**

Email (string): User's email address.

**Response:**

200 OK: Password reset link sent successfully.

*{*

*"message": "Password reset link has been sent to your mail"*

*}*

404 Not Found: User not found.

*{*

*"message": "User not found"*

*}*

500 Internal Server Error: Database error.

*{*

*"message": "Database error",*

*"error": "error\_details\_here"*

*}*

* **Reset Password (/user/reset\_password/<token>)**
  + **Method:** GET
  + Renders the password reset page.

**URL Parameters:**

token (string): Reset token received via email.

**Response:**

200 OK: Success.

*Renders the password reset page.*

400 Bad Request: Invalid or expired token.

*{*

*"message": "Invalid or expired token"*

*}*

* **Update Password (/user/update\_password)**
  + **Method:** POST
  + Expects to set a new password
  + Updates the user's password using the reset token.

**Form Data:**

token (string): Reset token received via email.

password (string): New password.

confirm\_password (string): Confirm new password.

**Response:**

200 OK: Password updated successfully.

*{*

*"message": "Password has been updated successfully"*

*}*

400 Bad Request: Passwords do not match or invalid token.

*{*

*"message": "Passwords do not match"*

*}*

500 Internal Server Error: Database error.

*{*

*"message": "Database error",*

*"error": "error\_details\_here"*

*}*

* **Change Password (/user/change\_password)**
  + **Method:** POST
  + Expects to provide old and new passwords
  + Changes the user's password.

**Headers:**

**Authorization (string):** JWT token.

**Request Body:**

old\_password (string): User's old password.

new\_password (string): User's new password.

confirm\_password (string): Confirm new password.

**Response:**

200 OK: Password changed successfully.

*{*

*"message": "Password has been changed successfully"*

*}*

400 Bad Request: Incorrect old password or missing fields.

*{*

*"message": "Incorrect old password"*

*}*

401 Unauthorized: Token is missing or invalid.

*{*

*"message": "Token is missing!"*

*}*

500 Internal Server Error: Database error.

*{*

*"message": "Database error",*

*"error": "error\_details\_here"*

*}*