API Reference and Documentation

Document Version: 1.0
Date: April 10, 2025

Author: ***George Udonkwo***

Mat no.: ***G2024/Meng/ETI/SET/FT/010***

Contents

API Reference Document

Overview

This API is designed to handle user authentication for a simple login system. It follows RESTful conventions and uses HTTP methods such as POST, GET, and DELETE. All requests and responses are formatted in JSON.

Base URL:

The base url for the login application is**:**

https://localhost:5000/api

Endpoints

The following endpoints are supported in the application

1. POST /auth/register

- Description: Register new users.
- Request Body (JSON):
- Example;

  {

  "username": "george",

  "password": "geo@sde809"

  }
- Response:
  - Success (200 OK):

  {

  "message": "Login successful",

  "token": "jwtOrOtherAuthToken"

  }
  - Failure (401 Unauthorized):

  {

  "error": "Invalid username or password"

  }

2. POST /auth/login

- Description: Logs a registered in and generate authentication token.

- Response:
    - Success (200 OK):

      {

       "message": "Login successful"

      “token”: “authorization token”

      }
    - Failure (401 Unauthorized):

      {

       "error": "Invalid credentials"

      }

3. GET /users/profile

- Description: Retrieves the authenticated user’s profile information.
- Request Headers:
    - Authorization: Bearer <token>
- Response:
    - Success (200 OK):

      {

       "username": "george",

       "email": "enogeorgeudonkwo@gmail.com",

      }
    - Failure (401 Unauthorized):

      {

       "error": "Unauthorized or expired token"

      }

Authentication

All secured endpoints require a valid authorization token. Include the token in the request header as follows:

Authorization: Bearer <token>

Error Handling and Status Codes

Status Code	Meaning	Example Cause

200	OK	Request completed successfully
201	Created	Resource successfully created
400	Bad Request	Missing or invalid request parameters
401	Unauthorized	Missing or invalid credentials
403	Forbidden	Authenticated but insufficient permissions
404	Not Found	Requested resource not found
500	Internal Server Error	Unhandled exception or server-side error

Documentation and Best Practices

Purpose

The primary purpose of the online login module is to securely authenticate users, ensuring that only authorized individuals gain access to protected resources and functionalities. This module is designed to safeguard sensitive data by verifying user credentials through robust authentication mechanisms. It aims to provide:

- User Authentication: Reliable verification of user identity using secure methods (e.g., username/password combinations).
- Security Assurance: Prevention of unauthorized access through encryption, token-based session management, and protection against common vulnerabilities like brute force attacks.
- User Experience: A streamlined, user-friendly process for logging in and gaining access, thereby reducing friction while maintaining strong security standard

Scope

The online login module encompasses a defined set of functionalities that form the entry point for user interaction with the system. Its scope includes, but is not limited to, the following elements:

- User Credential Management:
  - Login Functionality: Validate and process user credentials.
  - Password Management: Support secure password entry, storage, and transmission using encryption techniques.
- Session Management:
  - Token Generation and Validation: Issue secure session tokens (e.g., JWT) post-authentication and validate them for subsequent requests.
  - Session Expiration and Logout: Manage session timeouts and provide mechanisms for secure user logout.
- Error Handling and Security Logging:

    - Feedback Mechanisms: Provide clear error messages for failed authentication attempts.
- Integration Points:
    - API Endpoints: Offer RESTful endpoints for registration, login, and user profile retrieval
    - Integration with Database: Interface with back-end user databases for credential verification.

Setup Guide

Pre-requisites

Before proceeding, ensure you have the following installed on your system:

- **Python (version 3.7 or later)**: Verify your installation by running:
- **pip**: Python's package installer, which should be installed alongside Python.
- **Git (optional)**: For cloning the project repository.

Step 1: Clone the Repository

If you're starting from a Git repository, clone it to your local machine:

git clone https://github.com/Georgeudonkwo/login-module.git

Step 2: Create a Virtual Environment

A virtual environment isolates your project's dependencies from other projects. To create one:

- **On Windows:**

python -m venv venv

Step 3: Activate the Virtual Environment

Activate the virtual environment using the appropriate command:

- **On Windows:**

  venv\Scripts\activate

Step 4: Install Dependencies

With the virtual environment active, install the required dependencies by executing:

pip install -r requirements.txt

Usage Instructions

1. Starting the Server Locally

1. **Activate the Virtual Environment**
   - **Windows:** venv\Scripts\activate.
2. **Run the Server**
   The project entry point main.py, start the server using: **python main.p**y

   flask run, since the project is a flask application.

3. **Confirm the Server is Running**
   - Open your browser and navigate to http://localhost:5000 (adjust the port if necessary)

2. Testing Endpoints in Swagger UI

The Project project includes Swagger/OpenAPI documentation:

1. **Access Swagger UI**
   - Open your web browser and navigate to the dedicated Swagger URL http://localhost:5000/apidocs/.
2. **Explore API Endpoints**
   - Swagger UI displays a list of all available endpoints along with their supported HTTP methods, expected request parameters, and response schemas.
   - You can interact with each endpoint by:
     - Expanding the endpoint section.
     - Clicking the "Try it out" button.
     - Filling in any required parameters (such as authentication headers or request bodies).
     - Clicking "Execute" to send the request.
3. **Review API Responses**
   - Swagger UI displays response details including status codes, headers, and response bodies. This helps verify whether each endpoint is functioning as expected.

3. Testing Endpoints in Postman

Postman offers a comprehensive interface for API testing. Follow these steps to test your endpoints:

1. **Manual Setup of Requests**
    - **Creating a New Request:**
        - Click the "New" button and select "Request".
        - Enter the request name and assign it to a collection.
        - Specify the HTTP method (e.g., POST, GET, DELETE) and the request URL (e.g., http://localhost:5000/auth/login).
    - **Setting Request Headers:**
        - For secured endpoints, add the Authorization header. Example: Authorization: Bearer <your_token>
    - **Defining the Request Body:**
        - If your endpoint expects a JSON payload (e.g., login credentials), select the "Body" tab.
        - Choose "raw" and set the type to "JSON".
        - Enter the JSON payload. For example, for login:

          {

           "username": "username",

           "password": "password"

          }

2. **Sending the Request and Reviewing the Response**
    - Click "Send" to execute the request.
    - Postman displays the response, including status code, response time, and the response body.
    - Verify that the server responds as defined in your API documentation (e.g., a 200 OK for a successful login or appropriate error codes for invalid requests).

License Information:

The application is licensed under the MIT standard

Security Best Practices

- **Secure Storage of Secrets:** Store API keys, database credentials, etc., in environment variables.
- **Password Encryption:** Hash and salt passwords using werkzeug secure algorithms

Versioning the API

- **API Versioning:** Implement a versioning strategy (e.g., `/api/auth/v1/login`) to maintain backward compatibility.

Documentation Standards

- **Inline Code Documentation:** Use docstring annotations for function documentations.

- **API Specification Formats:** uses OpenAPI/Swagger standard to create an API specification for client generation.