**Environment Setup:**

* **Dependencies:**
  + Flask: Web framework for building APIs.
  + Flask-JWT-Extended: Library for handling JWT authentication.
  + Flask-CORS: Allows Cross-Origin Resource Sharing.
  + psycopg2: PostgreSQL database adapter.
  + python-dotenv: Loads environment variables from a .env file.
  + werkzeug: Provides utilities like password hashing.
  + input\_validator: Custom module to validate input data (e.g., checks for valid numbers or points).
* **Environment Variables:** The following environment variables must be set in a .env file:
  + DB\_HOST: PostgreSQL host address.
  + DB\_PORT: PostgreSQL port.
  + DB\_USER: PostgreSQL username.
  + DB\_PASSWORD: PostgreSQL password.
  + DB\_NAME: PostgreSQL database name.

**Application Structure:**

* **Routes:**
  + /: Home route, returns a welcome message.
  + /api/users: Fetches all users from the database.
  + /api/signup: Handles user sign-up by accepting username, password, subscribe, and email.
  + /api/login: Handles user login, returns a JWT token if credentials are valid.
  + /api/subscribed\_users: Fetches users who are subscribed.
  + /api/logout: Logs out a user by adding the JWT to a blacklist.
  + /api/protected: A protected route that requires JWT authentication.
  + /lesson/<int:id>: Fetches a lesson by its id.
  + /get\_total\_lessons: Returns the total count of lessons.
  + /tips/<int:id>: Fetches tips for a given id.
  + /get\_score/<int:userid>: Retrieves the total points of a user identified by userid.
  + /update\_points: Updates points for a specific user and topic.
  + /get\_lesson\_status/<int:userid>/<int:topicid>: Retrieves the completion status of a lesson for a user.
  + /complete\_lesson: Marks a lesson as completed for a user.

**Detailed Route Explanation:**

**1. Home Route**

* **Route**: /
* **Method**: GET
* **Description**: A simple route that returns a welcome message.
* **Response**:

json

{

"message": "Welcome to the homepage!"

}

**2. Get All Users**

* **Route**: /api/users
* **Method**: GET
* **Description**: Fetches all users from the database.
* **Response**:

json

[

{

"id": 1,

"username": "user1",

"password": "hashed\_password",

"admin": false,

"subscribe": true,

"email": "user1@example.com"

},

...

]

**3. User Signup**

* **Route**: /api/signup
* **Method**: POST
* **Description**: Handles user sign-up. Validates that the username is unique and stores the new user's data.
* **Request Body**:

json

{

"username": "newuser",

"password": "password123",

"subscribe": true,

"email": "newuser@example.com"

}

* **Response**:

json

{

"msg": "Signup successful",

"user\_id": 1

}

**4. User Login**

* **Route**: /api/login
* **Method**: POST
* **Description**: Authenticates a user based on username and password. If successful, it returns a JWT token.
* **Request Body**:

json

{

"username": "existinguser",

"password": "password123"

}

* **Response**:

json

{

"access\_token": "<JWT\_TOKEN>",

"userid": 1

}

**5. Get Subscribed Users**

* **Route**: /api/subscribed\_users
* **Method**: GET
* **Description**: Fetches a list of users who are subscribed.
* **Response**:

json

[

{

"username": "user1",

"email": "user1@example.com"

},

...

]

**6. User Logout**

* **Route**: /api/logout
* **Method**: POST
* **Description**: Logs out the user by adding the JWT to a blacklist.
* **Response**:

json

{

"msg": "Successfully logged out"

}

**7. Protected Route**

* **Route**: /api/protected
* **Method**: GET
* **Description**: A protected route that only allows access if the user is authenticated via JWT.
* **Response**:

json

{

"logged\_in\_as": "user\_id"

}

**8. Get Lesson by ID**

* **Route**: /lesson/<int:id>
* **Method**: GET
* **Description**: Fetches lesson details by id.
* **Response**:

json

{

"topicId": 1,

"topicName": "Lesson 1",

"topicContent": "Content of lesson 1",

"isInteractive": true

}

**9. Get Total Lessons**

* **Route**: /get\_total\_lessons
* **Method**: GET
* **Description**: Returns the total count of lessons.
* **Response**:

json

{

"total\_lessons": 5

}

**10. Get Tips for a Topic**

* **Route**: /tips/<int:id>
* **Method**: GET
* **Description**: Fetches tips associated with a given lesson id.
* **Response**:

json

[

{

"tipId": 1,

"topicId": 1,

"tipContent": "Tip content here",

"topicName": "Lesson 1"

},

...

]

**11. Get Total Points for User**

* **Route**: /get\_score/<int:userid>
* **Method**: GET
* **Description**: Fetches the total points accumulated by a user.
* **Response**:

json

{

"total\_points": 100

}

**12. Update Points for a User**

* **Route**: /update\_points
* **Method**: POST
* **Description**: Updates points for a specific user and topic.
* **Request Body**:

json

{

"userid": 1,

"topicid": 1,

"points": 10

}

* **Response**:

json

{

"message": "Points updated successfully for userid = 1"

}

**13. Get Lesson Completion Status**

* **Route**: /get\_lesson\_status/<int:userid>/<int:topicid>
* **Method**: GET
* **Description**: Retrieves the completion status for a specific lesson.
* **Response**:

json

{

"lesson\_status": true

}

**14. Complete Lesson**

* **Route**: /complete\_lesson
* **Method**: POST
* **Description**: Marks a lesson as completed for a user.
* **Request Body**:

json

{

"userid": 1,

"topicid": 1

}

* **Response**:

json

{

"message": "Lesson status updated successfully for userid = 1"

}

**Error Handling:**

* Each route handles exceptions using try-except blocks, returning a 500 status code with a descriptive error message when an error occurs.
* Common errors include:
  + Invalid input data (e.g., non-numeric IDs).
  + Missing or invalid parameters.
  + Database connection errors.

**JWT Authentication:**

* The application uses JWT tokens for secure authentication.
* The /api/protected and other JWT-protected routes require valid JWT tokens, which must be passed in the Authorization header as Bearer <token>.

**Database Setup:**

The application assumes the following database schema:

* **Users Table (users)**:
  + userid: Primary key (integer).
  + username: User's username (string).
  + password: User's hashed password (string).
  + admin: Boolean indicating if the user is an admin.
  + subscribe: Boolean indicating if the user is subscribed.
  + email: User's email address.
* **Lessons Table (lessons)**:
  + topicId: Primary key (integer).
  + topicName: Name of the lesson (string).
  + topicContent: Content of the lesson (text