Info Sec Management API

Overview

This is a RESTful API for user authentication and product management. It includes JWT-based authentication and CRUD operations for users and products.

Features

- User authentication with JWT
- CRUD operations for users and products
- Secure password handling with Bcrypt
- Database connection using SQLAlchemy

Technologies Used

- Flask
- Flask-SQLAlchemy
- Flask-JWT-Extended
- Flask-Bcrypt
- PyMySQL

Database Setup

1. Create the MySQL Database

Before running the application, create a MySQL database:

```
CREATE DATABASE info_sec_mgmt;
```

If you are using a MySQL user with a password, ensure you update `app.py` accordingly.

Installation & Setup

1. Create a Virtual Environment

```
python -m venv venv
venv\Scripts\activate  # Windows
```

2. Install Dependencies

```
pip install flask flask-sqlalchemy flask-jwt-extended flask-bcrypt
pymysql
```

3. Configure Database & Secret Key (Without .env)

Edit `app.py` and manually set these values:

```
app.config['SQLALCHEMY_DATABASE_URI'] =
'mysql+pymysql://root:@localhost/info_sec_mgmt'
app.config['JWT_SECRET_KEY'] = 'your_secret_key'
```

4. Run the Application

```
python app.py
```

The server will start on `http://127.0.0.1:5000/`

API Endpoints

Authentication

- **POST /signup** → Register a new user
- **POST /login** → Authenticate and receive JWT token

User Operations (Protected)

- **GET /users** → Retrieve all users
- **GET /users/{id}** → Retrieve a user by ID
- **DELETE /users/{id}** → Delete a user
- **PUT /users/{id}** → Update user details

Product Operations (Protected)

- **POST /products** → Add a new product
- **GET /products** → Retrieve all products
- **GET /products/{pid}** → Retrieve a product by ID
- **PUT /products/{pid}** → Update a product
- **DELETE /products/{pid}** → Delete a product

Testing with PowerShell

1. Register a New User

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/signup" -Method POST -
Headers @{"Content-Type"="application/json"} -Body '{"name": "Hager",
"username": "hager123", "password": "mypassword"}'
```

2. Login and Get JWT Token

```
$loginResponse = Invoke-WebRequest -Uri "http://127.0.0.1:5000/login"
-Method POST -Headers @{"Content-Type"="application/json"} -Body
'{"username": "hager123", "password": "mypassword"}'
$token = ($loginResponse.Content | ConvertFrom-Json).token
Write-Output "Your JWT Token: $token"
```

3. Add a Product

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/products" -Method POST -
Headers @{"Content-Type"="application/json"; "Authorization"="Bearer
$token"} -Body '{"pname": "Laptop", "description": "Gaming laptop",
"price": 1200.99, "stock": 10}'
```

4. Retrieve All Products

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/products" -Method GET -
Headers @{"Authorization"="Bearer $token"}
```

5. Retrieve a Specific Product

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/products/1" -Method GET
-Headers @{"Authorization"="Bearer $token"}
```

6. Update a Product

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/products/1" -Method PUT
-Headers @{"Content-Type"="application/json"; "Authorization"="Bearer
$token"} -Body '{"pname": "Updated Laptop", "description": "High-end
gaming laptop", "price": 1500.00, "stock": 5}'
```

7. Delete a Product

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/products/1" -Method
DELETE -Headers @{"Authorization"="Bearer $token"}
```

Security & Best Practices

- Passwords are hashed before storage
- JWT tokens are used for authentication
- Routes are protected with token validation

License

MIT License