Restful API And Microservices with Python

Day 8

Day 8 - Overview

- Role Based Access Control(RBCA)
- Update User Table to add Role column
- Creating initial Admin user on startup
- RBAC using jwt_claims
- Restricting Access to APIs based on roles
- User Schema Role validation Task

Prerequisite

- VM with windows OS
- Python 3.8 or >
- Visual Studio Code Code Editor
- Postman
- GIT

https://github.com/saurav-samantray/flask-microservices-training/blob/main/slides/Setup%20GIT% 20in%20your%20Local%20system.pdf

Docker - Not Mandatory for current training

Sync your fork for Day 8 activities

Follow the below document to sync your fork and update local repository.

https://github.com/saurav-samantray/flask-microservices-training/blob/main/slides/Setup%20GIT%20in%20your%20Local%20system.pdf

Local Setup for Day 8

Navigate to the below folder

C:\workspace\flask-microservices-training\day8\user-management-service

Create a virtual environment and activate it

python -m venv venv

.\venv\Scripts\activate

• Install the dependencies, initialize DB and start server

pip install -r requirements.txt

python init_db.py

python server.py

Role Based Access Control - RBAC

- Role-based access control (RBAC) refers to the idea of assigning permissions to users based on their role within an organization. It offers a simple, manageable approach to access management that is less prone to error than assigning permissions to users individually.
- For example, if you were using RBAC to control access for an HR application, you could give HR managers a role that allows them to update employee details, while other employees would be able to view only their own details.

Update User Table to add Role column

```
CREATE TABLE user
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   name TEXT NOT NULL,
   email TEXT NOT NULL UNIQUE,
   age INTEGER NOT NULL,
   password TEXT NOT NULL,
   role TEXT NOT NULL DEFAULT 'USER',
   created TIMESTAMP NOT NULL DEFAULT CURRENT TIMESTAME
```

Creating Admin User on startup

```
@app.before first request
def before first request():
    print("before first request")
    initial admin user = User.from json({
        'name': 'Admin User',
        'email': app.config['UMS ADMIN EMAIL'],
        'password': app.config['UMS ADMIN PASSWORD'],
        'role': 'ADMIN',
        'age': 45
    create admin user (flask bcrypt, initial admin user)
```

Custom Decorator to validate jwt claim

```
def admin required():
   def wrapper(fn):
        @wraps(fn)
        def decorator(*args, **kwargs):
            verify jwt in request()
            claims = get jwt()
            if claims["role"] == 'ADMIN':
                return fn(*args, **kwargs)
            else:
                return {'msg': "Admins only!"}, 403
        return decorator
   return wrapper
```

Q and A