### Step 1: Setting up the environment

### pip install Flask Flask-RESTful Flask-SQLAlchemy Flask-JWT-Extended

### Step 2: Setting up the Flask Application

Create a new file app.py and set up the Flask application along with the database configuration.

### from flask import Flask

### from flask\_restful import Api

### from flask\_sqlalchemy import SQLAlchemy

### from flask\_jwt\_extended import JWTManager

### app = Flask(\_\_name\_\_)

### api = Api(app)

### app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///data.db'

### app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

### app.config['JWT\_SECRET\_KEY'] = 'super-secret' # Change this to a random secret key

### db = SQLAlchemy(app)

### jwt = JWTManager(app)

### @app.before\_first\_request

### def create\_tables():

### db.create\_all()

### if \_\_name\_\_ == '\_\_main\_\_':

### app.run(debug=True)

### Step 3: Creating Models

Create a models.py file to define the database models for users and items.

### from app import db

### class User(db.Model):

### id = db.Column(db.Integer, primary\_key=True)

### username = db.Column(db.String(80), unique=True, nullable=False)

### password = db.Column(db.String(80), nullable=False)

### class Item(db.Model):

### id = db.Column(db.Integer, primary\_key=True)

### name = db.Column(db.String(80), unique=True, nullable=False)

### price = db.Column(db.Float, nullable=False)

### Step 4: Creating Resource Endpoints

Create a resources.py file to define the RESTful API endpoints for user authentication and item CRUD operations.

from flask\_restful import Resource, reqparse

from flask\_jwt\_extended import create\_access\_token, jwt\_required, get\_jwt\_identity

from werkzeug.security import safe\_str\_cmp

from models import User, Item

from app import db

class UserRegister(Resource):

#### parser = reqparse.RequestParser()

#### parser.add\_argument('username', type=str, required=True, help="This field cannot be blank.")

#### parser.add\_argument('password', type=str, required=True, help="This field cannot be blank.")

#### def post(self):

#### data = UserRegister.parser.parse\_args()

#### if User.query.filter\_by(username=data['username']).first():

#### return {'message': 'User already exists'}, 400

#### new\_user = User(username=data['username'], password=data['password'])

#### db.session.add(new\_user)

#### db.session.commit()

#### return {'message': 'User created successfully.'}, 201

#### # User Login Resource

#### class UserLogin(Resource):

#### parser = reqparse.RequestParser()

#### parser.add\_argument('username', type=str, required=True, help="This field cannot be blank.")

#### parser.add\_argument('password', type=str, required=True, help="This field cannot be blank.")

#### def post(self):

#### data = UserLogin.parser.parse\_args()

#### user = User.query.filter\_by(username=data['username']).first()

#### if user and safe\_str\_cmp(user.password, data['password']):

#### access\_token = create\_access\_token(identity=user.id)

#### return {'access\_token': access\_token}, 200

#### return {'message': 'Invalid credentials'}, 401

#### # Item Resource

#### class ItemResource(Resource):

#### parser = reqparse.RequestParser()

#### parser.add\_argument('price', type=float, required=True, help="This field cannot be blank.")

#### @jwt\_required()

#### def get(self, name):

#### item = Item.query.filter\_by(name=name).first()

#### if item:

#### return {'name': item.name, 'price': item.price}, 200

#### return {'message': 'Item not found'}, 404

#### @jwt\_required()

#### def post(self, name):

#### if Item.query.filter\_by(name=name).first():

#### return {'message': f"An item with name '{name}' already exists."}, 400

#### data = ItemResource.parser.parse\_args()

#### new\_item = Item(name=name, price=data['price'])

#### db.session.add(new\_item)

#### db.session.commit()

#### return {'name': new\_item.name, 'price': new\_item.price}, 201

#### @jwt\_required()

#### def put(self, name):

#### data = ItemResource.parser.parse\_args()

#### item = Item.query.filter\_by(name=name).first()

#### if item:

#### item.price = data['price']

#### else:

#### item = Item(name=name, price=data['price'])

#### db.session.add(item)

#### db.session.commit()

#### return {'name': item.name, 'price': item.price}, 200

#### @jwt\_required()

#### def delete(self, name):

#### item = Item.query.filter\_by(name=name).first()

#### if item:

#### db.session.delete(item)

#### db.session.commit()

#### return {'message': 'Item deleted'}, 200

#### return {'message': 'Item not found'}, 404

### Step 5: Adding Endpoints to the API

Update app.py to add the resource endpoints.

### from resources import UserRegister, UserLogin, ItemResource

### api.add\_resource(UserRegister, '/register')

### api.add\_resource(UserLogin, '/login')

### api.add\_resource(ItemResource, '/item/<string:name>')

### if \_\_name\_\_ == '\_\_main\_\_':

### app.run(debug=True)

### Step 6: Running the Application

Run the Flask application.

#### python app.py

**Testing the API**

**Register a new user:**

curl -X POST -H "Content-Type: application/json" -d '{"username":"user1", "password":"pass"}' <http://127.0.0.1:5000/register>

**Login with the user:**

curl -X POST -H "Content-Type: application/json" -d '{"username":"user1", "password":"pass"}' <http://127.0.0.1:5000/login>

**Create a new item:**

curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer <access\_token>" -d '{"price": 10.99}' <http://127.0.0.1:5000/item/item1>

**Get the item:**

curl -X GET -H "Content-Type: application/json" -H "Authorization: Bearer <access\_token>" <http://127.0.0.1:5000/item/item1>

**Update the item:**

curl -X PUT -H "Content-Type: application/json" -H "Authorization: Bearer <access\_token>" -d '{"price": 15.99}' <http://127.0.0.1:5000/item/item1>

**Delete the item:**

curl -X DELETE -H "Content-Type: application/json" -H "Authorization: Bearer <access\_token>" <http://127.0.0.1:5000/item/item1>