**app.py**

from flask import Flask, request

from flask\_restful import Api, Resource

from werkzeug.security import generate\_password\_hash, check\_password\_hash

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

api = Api(app)

users = {}

class Register\_User(Resource):

    def post(self):

        data = request.get\_json()

        username = data.get('username')

        password = data.get('password')

        if username in users:

            return {"notification": "User registered"}, 400

        hashed\_password = generate\_password\_hash(password)

        users[username] = hashed\_password

        return {"notification": "User created successfully"}, 201

class Login\_User(Resource):

    def post(self):

        data = request.get\_json()

        username = data.get('username')

        password = data.get('password')

        if username not in users or not check\_password\_hash(users[username], password):

            return {"notification": "Invalid credentials"}, 401

        return {"access\_token": "fake-token-for-{}".format(username)}, 200

class Resource\_Item(Resource):

    def get(self, item\_id):

        item = {"item\_id": item\_id, "name": "Test Item"}

        return item, 200

api.add\_resource(Register\_User, '/register')

api.add\_resource(Login\_User, '/login')

api.add\_resource(Resource\_Item, '/items/<int:item\_id>')

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

    app.run(debug=True)

**config.py**

import os

class TestApp\_Config:

    SECRET\_KEY = os.environ.get('SECRET\_KEY') or 'secret not disclosed'

    SQLALCHEMY\_DATABASE\_URI = 'sqlite:///db.sqlite3'

    SQLALCHEMY\_TRACK\_MODIFICATIONS = False

    JWT\_SECRET\_KEY = 'secret kept hidden'

**models.py**

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.security import generate\_password\_hash, check\_password\_hash

db = SQLAlchemy()

class TestUser(db.Model):

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

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

    password\_hash = db.Column(db.String(128), nullable=False)

    def keep\_password(self, password):

        self.password\_hash = generate\_password\_hash(password)

    def analyze\_password(self, password):

        return check\_password\_hash(self.password\_hash, password)

class TestItem(db.Model):

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

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

    description = db.Column(db.String(200), nullable=True)

**resources.py**

from flask import request

from flask\_restful import Resource

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

from models import db, User, Item

class Register\_User(Resource):

    def post(self):

        data = request.get\_json()

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

            return {'notification': 'User created'}, 400

        user = User(username=data['username'])

        user.set\_password(data['password'])

        db.session.add(user)

        db.session.commit()

        return {'notification': 'User registered successfully'}, 201

class Login\_User(Resource):

    def post(self):

        data = request.get\_json()

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

        if user and user.check\_password(data['password']):

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

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

        return {'notification': 'Invalid credentials'}, 401

class Resource\_Item(Resource):

    @jwt\_required()

    def get(self, item\_id):

        item = Item.query.get\_or\_404(item\_id)

        return {'id': item.id, 'name': item.name, 'description': item.description}

    @jwt\_required()

    def delete(self, item\_id):

        item = Item.query.get\_or\_404(item\_id)

        db.session.delete(item)

        db.session.commit()

        return {'message': 'Item deleted'}

    @jwt\_required()

    def put(self, item\_id):

        putdata = request.get\_json()

        item = Item.query.get\_or\_404(item\_id)

        item.name = putdata['name']

        item.description = putdata['description']

        db.session.commit()

        return {'id': item.id, 'name': item.name, 'description': item.description}

class ShowItemListResource(Resource):

    @jwt\_required()

    def get(self):

        items = Item.checkquery.all()

        return [{'id': item.id, 'name': item.name, 'description': item.description} for item in items]

    @jwt\_required()

    def post(self):

        postdata = request.get\_json()

        item = Item(name=postdata['name'], description=postdata['description'])

        db.session.add(item)

        db.session.commit()

        return {'id': item.id, 'name': item.name, 'details': item.details}, 201