**Module – 5**

**Assignment 10: Build the Test API**

**Objective**

Create RESTful API endpoints for managing speaking test data in the IELTS Speaking Test platform, enabling CRUD (Create, Read, Update, Delete) operations on speaking test records.

Sure! Here is the Python code to create RESTful API endpoints for managing speaking test data using Flask and SQLAlchemy. This implementation includes the necessary routes to handle CRUD operations, validation and error handling.

**Step-by-Step Approach and Code**

**Step 1: Set Up API Routes**

- Create a `speaking\_tests` blueprint to manage all endpoints.

- Base URL: `/api/speaking-tests`.

**Step 2: Implement Endpoints**

- Define endpoints for:

- `POST /api/speaking-tests` to add a new speaking test record.

- `GET /api/speaking-tests/<int:test\_id>` to retrieve a specific test record by ID.

- `GET /api/speaking-tests` to retrieve all speaking test records.

- `PUT /api/speaking-tests/<int:test\_id>` to update an existing test record.

- `DELETE /api/speaking-tests/<int:test\_id>` to delete a test record by ID.

**Step 3: Validation and Error Handling**

- Validate request payloads for required fields.

- Handle errors such as invalid test IDs and missing data.

**Step 4: Integration with SQLAlchemy Models**

- Use the `SpeakingTest` model for database operations.

**Step 5: Testing**

- Test all endpoints using Postman or a similar tool.

**Code Implementation**

1. \*\*Create a Blueprint for the Endpoints\*\*

2. \*\*Implement the Endpoints with CRUD Operations\*\*

3. \*\*Add Error Handling and Validation\*\*

4. \*\*Register the Blueprint in the Main Application\*\*

**1. Create the Blueprint (`speaking\_tests.py`)**

```python

from flask import Blueprint, request, jsonify, abort

from models import db, SpeakingTest

speaking\_tests = Blueprint('speaking\_tests', \_\_name\_\_)

# Validation function to check required fields

def validate\_speaking\_test\_data(data):

if 'user\_id' not in data or 'question' not in data or 'response' not in data or 'score' not in data:

abort(400, description="Missing required fields")

# POST /api/speaking-tests

@speaking\_tests.route('/api/speaking-tests', methods=['POST'])

def add\_speaking\_test():

data = request.get\_json()

validate\_speaking\_test\_data(data)

new\_test = SpeakingTest(user\_id=data['user\_id'], question=data['question'], response=data['response'], score=data['score'])

db.session.add(new\_test)

db.session.commit()

return jsonify({"message": "Speaking test added", "test": new\_test.id}), 201

# GET /api/speaking-tests/<int:test\_id>

@speaking\_tests.route('/api/speaking-tests/<int:test\_id>', methods=['GET'])

def get\_speaking\_test(test\_id):

test = SpeakingTest.query.get\_or\_404(test\_id)

return jsonify(test.to\_dict())

# GET /api/speaking-tests

@speaking\_tests.route('/api/speaking-tests', methods=['GET'])

def get\_all\_speaking\_tests():

tests = SpeakingTest.query.all()

return jsonify([test.to\_dict() for test in tests])

# PUT /api/speaking-tests/<int:test\_id>

@speaking\_tests.route('/api/speaking-tests/<int:test\_id>', methods=['PUT'])

def update\_speaking\_test(test\_id):

test = SpeakingTest.query.get\_or\_404(test\_id)

data = request.get\_json()

validate\_speaking\_test\_data(data)

test.user\_id = data['user\_id']

test.question = data['question']

test.response = data['response']

test.score = data['score']

db.session.commit()

return jsonify({"message": "Speaking test updated", "test": test.to\_dict()})

# DELETE /api/speaking-tests/<int:test\_id>

@speaking\_tests.route('/api/speaking-tests/<int:test\_id>', methods=['DELETE'])

def delete\_speaking\_test(test\_id):

test = SpeakingTest.query.get\_or\_404(test\_id)

db.session.delete(test)

db.session.commit()

return jsonify({"message": "Speaking test deleted"})

```

**2. Update `app.py` to Include the Blueprint**

```python

from flask import Flask

from config import Config

from models import db

from speaking\_tests import speaking\_tests

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

app.config.from\_object(Config)

db.init\_app(app)

app.register\_blueprint(speaking\_tests)

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

with app.app\_context():

db.create\_all()

app.run(debug=True)

```

**3. Define `models.py` for SQLAlchemy Models**

```python

from flask\_sqlalchemy import SQLAlchemy

from datetime import datetime

db = SQLAlchemy()

class User(db.Model):

\_\_tablename\_\_ = 'users'

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

name = db.Column(db.String(100), nullable=False)

email = db.Column(db.String(100), unique=True, nullable=False)

password = db.Column(db.String(100), nullable=False)

speaking\_tests = db.relationship('SpeakingTest', backref='user', lazy=True)

listening\_tests = db.relationship('ListeningTest', backref='user', lazy=True)

class SpeakingTest(db.Model):

\_\_tablename\_\_ = 'speaking\_tests'

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

user\_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

question = db.Column(db.String(200), nullable=False)

response = db.Column(db.Text, nullable=False)

score = db.Column(db.Float)

timestamp = db.Column(db.DateTime, default=datetime.utcnow)

def to\_dict(self):

return {

'id': self.id,

'user\_id': self.user\_id,

'question': self.question,

'response': self.response,

'score': self.score,

'timestamp': self.timestamp.isoformat()

}

class ListeningTest(db.Model):

\_\_tablename\_\_ = 'listening\_tests'

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

user\_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

question = db.Column(db.String(200), nullable=False)

response = db.Column(db.Text, nullable=False)

score = db.Column(db.Float)

timestamp = db.Column(db.DateTime, default=datetime.utcnow)

```

**4. Configure Database Settings (`config.py`)**

```python

class Config:

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

SQLALCHEMY\_TRACK\_MODIFICATIONS = False

```

**Testing the Endpoints**

1. \*\*Use Postman or Curl to Test:\*\*

- \*\*POST /api/speaking-tests\*\*

- Add a new speaking test record.

- \*\*GET /api/speaking-tests/:test\_id\*\*

- Retrieve a specific test record by ID.

- \*\*GET /api/speaking-tests\*\*

- Retrieve all speaking test records.

- \*\*PUT /api/speaking-tests/:test\_id\*\*

- Update an existing test record.

- \*\*DELETE /api/speaking-tests/:test\_id\*\*

- Delete a test record by ID.

**Submission Guidelines**

1. \*\*Submit all modified files:\*\*

- `speaking\_tests.py`

- Any additional updates to the backend (e.g., `app.py`, `models.py`, `config.py`).

2. \*\*Include Testing Evidence:\*\*

- Provide screenshots or screen recordings showing successful API interactions using Postman or a similar tool.

**Evaluation Criteria**

1. \*\*Endpoint Functionality (40%):\*\*

- Ensure all endpoints work as expected.

2. \*\*Validation and Error Handling (30%):\*\*

- Proper validation and meaningful error messages.

3. \*\*Code Quality (20%):\*\*

- Clean, modular, and well-documented code.

4. \*\*Submission Completeness (10%):\*\*

- All required files and testing evidence included.