**Task 3- Overview**

This task involves integrating OpenAI API calls into **mcq.py** to generate 5 challenging multiple-choice questions on topic chosen in dropdown, complexity level from radio buttons and submit the questions and generate the feedback for all questions.

**Task List**

1. **Understand the Boilerplate Code**: Review the structure and logic for multiple-choice question (MCQ) generation and evaluation.
   * Explore the data models and flow of JSON requests and responses.
2. **Implement Prompt Formatting**: Write prompts for generating MCQs based on a topic and complexity level.
3. **Integrate OpenAI API**: Complete the OpenAI API integration to dynamically generate MCQs and evaluate user submissions.
4. **Test and Debug**: Validate the functionality of each endpoint, ensuring correct processing of JSON data.

**Task Solution**

Update boilerplate **mcq.py** code with two endpoints:

**Challenge 1: (**/mcq/generate)

1. Create a prompt, json\_schema and invoke OpenAI API to get the questions in following format. Output should be structured in json format.

{

    "Id": "Q1",

    "Question": "What is the capital of France?",

    "Options": [

        {

            "OptionIndex": 0,

            "OptionValue": "Berlin"

        },

        {

            "OptionIndex": 1,

            "OptionValue": "Madrid"

        },

        {

            "OptionIndex": 2,

            "OptionValue": "Paris"

        },

        {

            "OptionIndex": 3,

            "OptionValue": "Rome"

        }

    ],

    "CorrectOptionIndex": 2,

    "Complexity": "Basic"

}

**Challenge 2:**

**Update the application to use Python classes instead of Schema**

**Challenge 1 Solution**

    question\_schema = {

        "type": "object",

        "properties": {

            "Id": {

                "type": "string",

                "description": "Unique identifier for the question (e.g., Q1, Q2, etc.).",

            },

            "Question": {

                "type": "string",

                "description": "The text of the multiple-choice question.",

            },

            "Options": {

                "type": "array",

                "description": "An array of possible answer options.",

                "items": {

                    "type": "object",

                    "properties": {

                        "OptionIndex": {

                            "type": "integer",

                            "description": "The index of the option (0-based).",

                        },

                        "OptionValue": {

                            "type": "string",

                            "description": "The text of the answer option.",

                        },

                    },

                    "required": ["OptionIndex", "OptionValue"],

                    "additionalProperties": False,

                },

            },

            "CorrectOptionIndex": {

                "type": "integer",

                "description": "The index of the correct answer option (0-based).",

            },

            "Complexity": {

                "type": "string",

                "enum": ["Basic", "Intermediate", "Advanced"],

                "description": "The complexity level of the question.",

            },

        },

        "required": ["Id", "Question", "Options", "CorrectOptionIndex", "Complexity"],

        "additionalProperties": False,

    }

**Challenge 2 Solution**

class Option(BaseModel):

    OptionIndex: int = Field(..., description="The index of the option (0-based).")

    OptionValue: str = Field(..., description="The text of the answer option.")

class QuestionModel(BaseModel):

    Id: str = Field(..., description="Unique identifier for the question (e.g., Q1, Q2, etc.).")

    Question: str = Field(..., description="The text of the multiple-choice question.")

    Options: List[Option] = Field(..., description="An array of possible answer options.")

    CorrectOptionIndex: int = Field(..., description="The index of the correct answer option (0-based).")

    Complexity: str = Field(..., description="The complexity level of the question.", enum=["Basic", "Intermediate", "Advanced"])

response = client.beta.chat.completions.parse(

        model="gpt-4o",

        response\_format=QuestionModel,

        messages=messages,

)