AI-Powered Quiz Generator Development Roadmap

This document outlines a step-by-step roadmap for building an AI-Powered Quiz Generator using Django and a pre-trained AI model. The platform will automatically generate quizzes from uploaded text documents or PDFs, utilizing NLP to extract key concepts and create multiple-choice questions, with features for difficulty level adjustment and performance tracking.

Phase 1: Setting Up the Foundation (Weeks 1-2)

Step 1: Project Initialization & Environment Setup

- **Install Python:** Ensure you have a recent version of Python (3.8+) installed on your system.
- **Set up a Virtual Environment:** Create a virtual environment using venv or conda to isolate project dependencies.
  ```
  python -m venv venv
  source venv/bin/activate  # On macOS/Linux
  .\venv\Scripts\activate  # On Windows
  ```
- **Install Django:** Install the Django framework within your virtual environment.
  ```
  pip install Django
  ```
- **Create a Django Project:** Start a new Django project.
  ```
  django-admin startproject quiz_generator
  cd quiz_generator
  ```
- **Create a Django App:** Create a dedicated Django app for your quiz functionality.
  ```
  python manage.py startapp quiz
  ```
- **Basic Project Configuration:**
  - Register your quiz app in quiz_generator/settings.py.
  - Configure your database (SQLite is fine for development).
  - Set up basic URL routing in quiz_generator/urls.py to include your app's URLs.

Step 2: Core Data Models

- **Define Models in quiz/models.py:** Design the database models to represent your data. Consider the following:
  - Document: Stores uploaded documents (title, upload date, file path/content).
  - Quiz: Represents a generated quiz (title, document it's based on, creation date).

- Question: Stores individual questions (quiz it belongs to, question text, correct answer, incorrect answers, difficulty level).
- UserProfile (if you plan user accounts): Stores user-specific data (e.g., performance history).
- QuizAttempt (if you plan performance tracking): Stores user attempts at quizzes (user, quiz, start time, end time, score).

```
# quiz/models.py
from django.db import models
from django.contrib.auth.models import User

class Document(models.Model):
    title = models.CharField(max_length=255)
    uploaded_at = models.DateTimeField(auto_now_add=True)
    file = models.FileField(upload_to='documents/')  # For file uploads
    content = models.TextField(blank=True, null=True) # For text input

    def __str__(self):
        return self.title

class Quiz(models.Model):
    title = models.CharField(max_length=255)
    document = models.ForeignKey(Document, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class Question(models.Model):
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE,
related_name='questions')
    text = models.TextField()
    correct_answer = models.CharField(max_length=255)
    incorrect_answers = models.JSONField()  # Store as a list of strings
    difficulty = models.CharField(max_length=10, choices=[('easy', 'Easy'),
('medium', 'Medium'), ('hard', 'Hard')], default='medium')

    def __str__(self):
        return self.text

class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # Add any additional user-specific fields

    def __str__(self):
```

```
        return self.user.username

class QuizAttempt(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    quiz = models.ForeignKey(Quiz, on_delete=models.CASCADE)
    start_time = models.DateTimeField(auto_now_add=True)
    end_time = models.DateTimeField(null=True, blank=True)
    score = models.FloatField(default=0.0)

    def __str__(self):
        return f"{self.user.username} - {self.quiz.title}"
```

- **Make Migrations:** Create and apply database migrations to create the tables.
  python manage.py makemigrations quiz
  python manage.py migrate

Phase 2: Integrating the AI Model (Weeks 3-6)

Step 3: Choosing and Preparing Your Pre-trained AI Model

- **Identify a Suitable NLP Model:** Based on your needs for key concept extraction and question generation, consider models like:
  - **Sentence Transformers:** For embedding text and finding similar concepts.
  - **Language Models (e.g., T5, GPT-2/3/Neo):** Fine-tuned or used with prompting for question generation.
  - **Libraries like spaCy or NLTK:** For basic NLP tasks like tokenization, part-of-speech tagging, and named entity recognition, which can aid in concept extraction.
- **Installation:** Install the necessary libraries for your chosen model (e.g., transformers, torch, spacy).
  pip install transformers torch spacy
  python -m spacy download en_core_web_sm # Example for spaCy

- **Model Loading:** Write Python code to load your pre-trained model. This might involve using the transformers library or loading a saved model.
- **Text Preprocessing:** Implement functions to clean and preprocess the input text from uploaded documents (e.g., removing special characters, handling different file formats using libraries like PyPDF2 or python-docx).

Step 4: Implementing Concept Extraction

- **Develop Extraction Logic:** Use your chosen NLP model and techniques to

extract key concepts from the processed text. This could involve:
    - **Keyword/Keyphrase Extraction:** Identifying important words or phrases.
    - **Named Entity Recognition (NER):** Identifying entities like people, organizations, locations, and concepts.
    - **Topic Modeling:** Discovering underlying themes in the text.
    - **Sentence Embeddings:** Embedding sentences and finding those with high information content.
- **Integrate with Django:** Create a function or class within your quiz app (e.g., in a utils.py or ai_integration.py file) that takes text as input and returns a list of extracted concepts.

Step 5: Implementing Question Generation

- **Develop Question Generation Logic:** Based on the extracted concepts, design a process to generate multiple-choice questions. This might involve:
    - **Template-based Generation:** Using templates with placeholders for concepts and generating questions by filling them.
    - **Abstractive Question Generation:** Using language models to generate questions more creatively based on the context. This often requires more advanced prompting or fine-tuning.
    - **Distractor Generation:** Creating plausible incorrect answer choices. This can be done using techniques like finding similar concepts or using word embeddings.
- **Difficulty Level Assignment:** Implement logic to assign difficulty levels (easy, medium, hard) to the generated questions. This could be based on the complexity of the concept, the length of the question/answers, or the type of reasoning required.
- **Integrate with Django:** Create a function or class that takes extracted concepts (and potentially the original text) as input and returns a list of Question objects.

Step 6: Connecting AI with Django Views

- **Create Views:** Develop Django views to handle:
    - **Document Upload:** A view with a form to allow users to upload text files or PDFs (or input text directly).
    - **Quiz Generation:** A view that takes the uploaded document (or text), calls your concept extraction and question generation functions, and saves the generated Quiz and Question objects to the database. Display a confirmation or allow users to review/edit the quiz.
- **Create Forms:** Define Django forms in quiz/forms.py for document uploading.
- **Update URLs:** Configure URLs in quiz/urls.py to map the views to specific URL

patterns.
- **Create Templates:** Design HTML templates to render the document upload form and display the generated quiz.

Phase 3: User Interface and Functionality (Weeks 7-10)

Step 7: Building the User Interface

- **Design Templates:** Create user-friendly HTML templates for:
  - Displaying the generated quizzes.
  - Presenting questions to the user.
  - Handling user answers and submissions.
  - Displaying quiz results and performance.
- **Implement Styling:** Use CSS (and potentially a CSS framework like Bootstrap or Tailwind CSS) to style the application and make it visually appealing.
- **Add JavaScript (if needed):** Implement any client-side interactivity, such as handling answer selections or providing immediate feedback.

Step 8: Implementing Quiz Taking Functionality

- **Create Views:** Develop Django views to:
  - Display a specific quiz to the user.
  - Process user submissions (store answers, calculate scores).
  - Display the results of a quiz attempt.
- **Update URLs:** Add URLs to handle quiz viewing and submission.
- **Update Models (if needed):** Ensure your models (QuizAttempt) can store user responses and calculate scores.

Step 9: Implementing Performance Tracking

- **Update Views:** Modify your quiz submission view to:
  - Record the user's answers and the time taken.
  - Calculate the score for the quiz attempt.
  - Save the QuizAttempt object to the database.
- **Create Views for Performance Display:** Develop views to:
  - Display a user's quiz history.
  - Show detailed results for individual quiz attempts.
  - Potentially provide summary statistics (e.g., average score, performance by topic).
- **Update Templates:** Create templates to display performance data.

Step 10: User Authentication and Authorization (Optional but Recommended)

- **Enable Django's Authentication System:** Use Django's built-in user

authentication system to allow users to create accounts, log in, and log out.
- **Implement User Profiles:** If needed, extend the user model with a UserProfile to store additional user-specific information.
- **Implement Authorization:** Control access to certain features based on user roles or permissions (e.g., only logged-in users can take quizzes or view their history).

Phase 4: Refinement and Deployment (Weeks 11-Ongoing)

Step 11: Testing and Debugging

- **Write Unit Tests:** Create unit tests for your models, views, and AI integration logic to ensure they function correctly.
- **Perform Integration Tests:** Test the interaction between different parts of your application (e.g., document upload to quiz generation).
- **User Testing:** Have others test your application to identify usability issues and bugs.
- **Debug and Fix Issues:** Address any bugs or issues identified during testing.

Step 12: Deployment

- **Choose a Hosting Provider:** Select a platform to host your Django application (e.g., Heroku, PythonAnywhere, AWS, Google Cloud).
- **Configure Deployment Settings:** Adjust your Django settings for production (e.g., database configuration, static file handling, security settings).
- **Deploy Your Application:** Follow the deployment instructions provided by your chosen hosting provider.

Step 13: Continuous Improvement

- **Gather User Feedback:** Collect feedback from users to identify areas for improvement.
- **Monitor Performance:** Track the performance of your application and optimize as needed.
- **Iterate and Add Features:** Continuously improve your quiz generator by adding new features, refining the AI model, and enhancing the user experience.

Key Considerations:

- **AI Model Selection and Fine-tuning:** The choice of your pre-trained model will significantly impact the quality of your quiz generation. You might need to experiment with different models or even fine-tune a model on a relevant dataset for better performance.
- **Handling Different Document Formats:** Ensure your application can handle

various file types (text, PDF, DOCX) effectively.
- **Scalability:** Consider how your application will handle a growing number of users and documents.
- **Security:** Implement appropriate security measures to protect user data and prevent vulnerabilities.
- **Error Handling:** Implement robust error handling to gracefully manage unexpected situations.

This roadmap provides a comprehensive guide. The timeline for each phase can vary depending on your experience, the complexity of the AI model you choose, and the scope of features you implement. Good luck with your AI-Powered Quiz Generator!