Department of Computer Science & Information Technology

CT-361: Artificial Intelligence and Expert Systems

CCP REPORT

"Quiz Generation"

Group members:

- Emaaz Ur Rehman CT-22033
- Siddiq sheikh CT-22027
- Abeer Ahmed CT-22041

Tables of Content

Abstract

The Quiz Generation App is a full-stack web application that automates the creation of multiple-choice questions (MCQs) from PDF documents such as lecture slides and notes. Utilizing the advanced LLaMA 3.3 (NVIDIA) language model, it extracts key information and intelligently formulates quiz questions. Users can review, edit, or add questions manually, and export them as a Google Form. The solution provides educators and trainers with a time-saving, AI-assisted method for assessment preparation.

Problem Statement

Educators spend a significant amount of time converting lecture content into assessments, especially quizzes. Manual quiz creation is not only time-consuming but also prone to bias and repetition. There is a pressing need for an automated system that transforms lecture material into quality quiz questions, while still allowing human oversight and customization.

Objective

- Automate the generation of quiz questions from PDF lecture slides.
- Enable users to refine generated questions.
- Provide easy export functionality to widely used platforms like Google Forms.
- Maintain a responsive, user-friendly interface for a seamless experience.

Features

- **PDF Upload**: Upload PDFs containing lecture slides or text content.
- **AI-Powered MCQ Generation**: Use LLaMA 3.3 (NVIDIA) to generate relevant multiple-choice questions.
- **Question Management**: Edit, delete, or manually add questions.
- **Google Form Export**: Export the finalized quiz directly to Google Forms.

Workflow

1. User uploads PDF.
2. Backend parses text from the file.
3. Parsed text is sent to model for MCQ generation.

4. Generated MCQs are returned and displayed on the frontend.
5. User reviews, edits, or adds new questions.
6. User clicks "Quiz link" to push finalized questions to a Google Form.
7. User clicks "View Responses" to see the responses.

Python Backend Code Explanation

1. clean_text(text)

- **Purpose**: Cleans raw text extracted from the PDF by removing unnecessary whitespace and formatting inconsistencies.
- **Code**:

```
def clean_text(text):
    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces/newlines with a single space
    text = text.strip()  # Remove leading/trailing spaces
    return text
```

- **Justification**: Extracted PDF text often contains irregular spacing, line breaks, and formatting noise. This function ensures a normalized string for better model input, reducing hallucinations and improving MCQ quality.

2. format_text(text)

- **Purpose**: Adds basic structure to the cleaned text, such as recognizing bullet points and uppercase headings.
- **Code**:

```
def format_text(text):
    lines = text.split("\n")
    formatted_text = []
    for line in lines:
        line = line.strip()
        if line.startswith("•") or line.startswith("-"):  # Bullet points
            formatted_text.append(f"\n- {line[1:].strip()}")
        elif line.isupper():  # Assuming uppercase lines are headings
            formatted_text.append(f"\n\n{line}\n{'=' * len(line)}")
        else:
            formatted_text.append(line)
    return " ".join(formatted_text)
```

- **Justification**: Format consistency (like bullet points and headings) helps the AI model understand context better. This improves the relevance and clarity of the generated MCQs.

3. extract_text_from_pdf(file)

- **Purpose**: Asynchronously reads the uploaded PDF file, extracts all text, and formats it for MCQ generation.
- **Code**:

```python
async def extract_text_from_pdf(file: UploadFile):
    file_content = await file.read()  # Read file content
    try:
        doc = fitz.open(stream=file_content, filetype="pdf")  # Open PDF
        text = "\n".join([page.get_text("text") for page in doc])  # Extract text
        formatted_text = clean_text(text)  # Apply text cleaning
        formatted_text = format_text(formatted_text)
        return formatted_text
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error extracting text: {str(e)}")
```

- **Justification**: Combines text extraction and preprocessing in a single asynchronous function, improving performance in a web environment and reducing backend response latency.

4. Prompt Design

- **Purpose**: Ensures that the AI model receives structured instructions and returns only valid JSON.
- **Snippet**:

```
prompt = f""" [INST] <<SYS>>
IMPORTANT:You MUST respond with ONLY valid JSON in the EXACT format specified below.
DO NOT include any additional text, explanations, or markdown formatting.

Generate exactly {number} multiple-choice questions (MCQs) based on the following text.
Ensure that:
1. Each question is directly based on facts from the text.
2. Each question has exactly four options.
3. The correct answer is one of the four options.
4. Irrelevant or generic questions (e.g., about page numbers, authors) are NOT included.
5.Each question must have 4 options and one correct answer.
6. The correct answer must be from the given options.

Required JSON format:
{{
    "questions": [
        {{
            "question": "Your question here",
            "options": ["Option 1", "Option 2", "Option 3", "Option 4"],
            "correct_answer": "Correct option"
        }},
        // Repeat for {number} questions total
    ]
}}
<</SYS>>

Generate {number} MCQs using:
1. The provided text (for context)
2. Your own knowledge (to enhance questions if needed)
Text to base questions on:
{text}
[/INST]"""
```

- **Justification**:
    - Prevents unwanted formatting (like markdown or explanations).

- Improves automation reliability by instructing the model to return strict JSON format.

5. AI model usage

- **Purpose**: Sends the prompt to the LLaMA 3.3 model and receives generated content.

```
response = client.chat.completions.create(
    model=model,
    messages=[{"role": "user", "content": prompt}],
    temperature=0.3,
    top_p=0.9,
)
```

- **Justification**:
  - Ensures secure and structured communication with the model.
  - Includes headers to provide context for logging or platform-specific behaviors.
  - temperature=0.3 and top_p=0.9 control creativity and diversity while maintaining accuracy.

6. json.loads(mcq_data)

- **Purpose**: Converts the raw string from the AI model into a Python dictionary.
- **Justification**: Enables the backend to return structured JSON data to the frontend, which can then be displayed, edited, or exported.

8. Generate google form

- **Purpose**: Sends finalized quiz questions to a Google Apps Script endpoint, which creates a corresponding Google Form with those questions.
- **Code**:

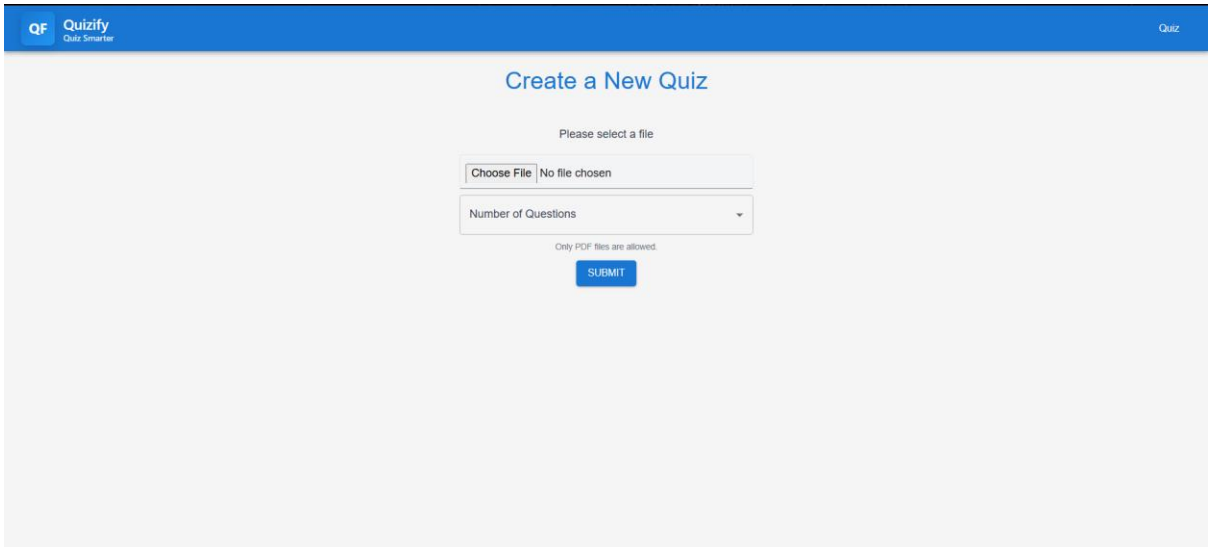
```
@app.post("/generate-form")
async def generate_form(quiz: QuizRequest):
    try:
        payload = {"questions": [q.dict() for q in quiz.questions]}

        response = requests.post(
            GOOGLE_SCRIPT_URL,
            json=payload,
            headers={'Content-Type': 'application/json'}
        )

        response.raise_for_status()

        return response.json()

    except requests.exceptions.RequestException as e:
        raise HTTPException(
            status_code=400,
            detail=f"Error communicating with Google Script: {str(e)}"
        )
    except Exception as e:
        raise HTTPException(
            status_code=500,
            detail=f"Internal server error: {str(e)}"
        )
```

- **Justification**:
    - This endpoint acts as a **bridge between the quiz generation system and Google Forms**.
    - It performs form creation from Google Apps Script
    - Google scripts code for generating Google form Link:

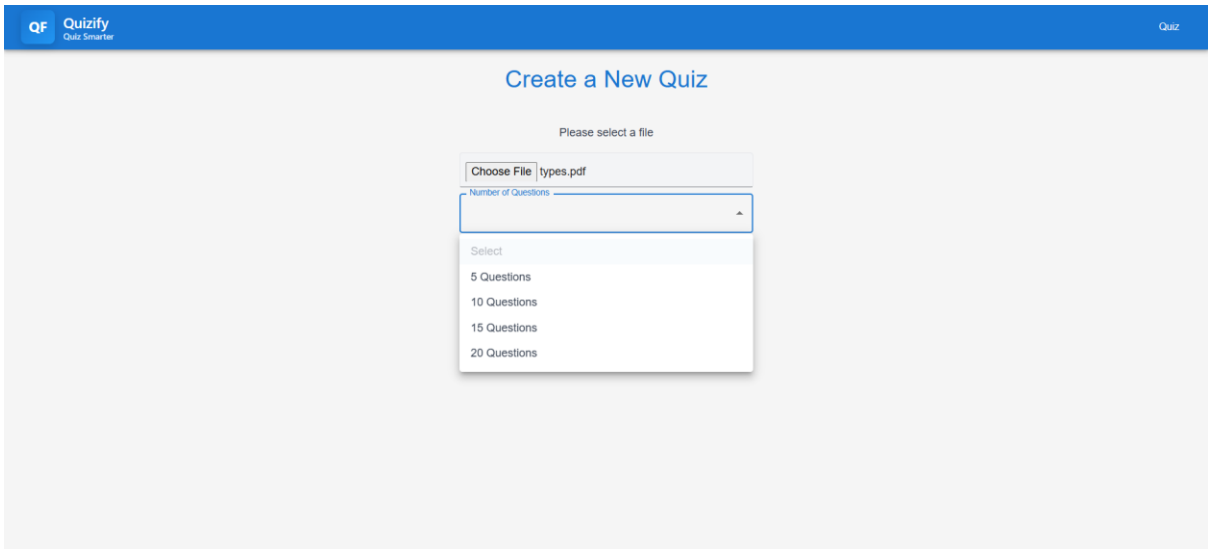
```
var formFile = DriveApp.getFileById(form.getId());
var sheet = SpreadsheetApp.create("Responses for " + form.getTitle());
form.setDestination(FormApp.DestinationType.SPREADSHEET, sheet.getId());
var sheetFile = DriveApp.getFileById(sheet.getId());

// Set sharing permissions - CRITICAL STEP
formFile.setSharing(DriveApp.Access.ANYONE_WITH_LINK, DriveApp.Permission.EDIT);
sheetFile.setSharing(DriveApp.Access.ANYONE_WITH_LINK, DriveApp.Permission.EDIT);

// If shareEmail is provided, explicitly add as editor
if (shareEmail) {
  try {
    formFile.addEditor(shareEmail);
    sheetFile.addEditor(shareEmail);

    // Add as owner if needed (requires domain admin rights)
    // formFile.setOwner(shareEmail); // Uncomment if you can transfer ownership
  } catch (e) {
    Logger.log("Error sharing with " + shareEmail + ": " + e.toString());
  }
}

// Generate URLs
var formUrl = form.getPublishedUrl();
var editUrl = `https://docs.google.com/forms/d/${form.getId()}/edit`;
var responseUrl = `https://docs.google.com/spreadsheets/d/${sheet.getId()}/edit`;

return {
  success: true,
  form_url: formUrl,
  edit_url: editUrl,
  response_url: responseUrl,
  message: shareEmail ? "Shared with " + shareEmail : "No specific user shared"
```

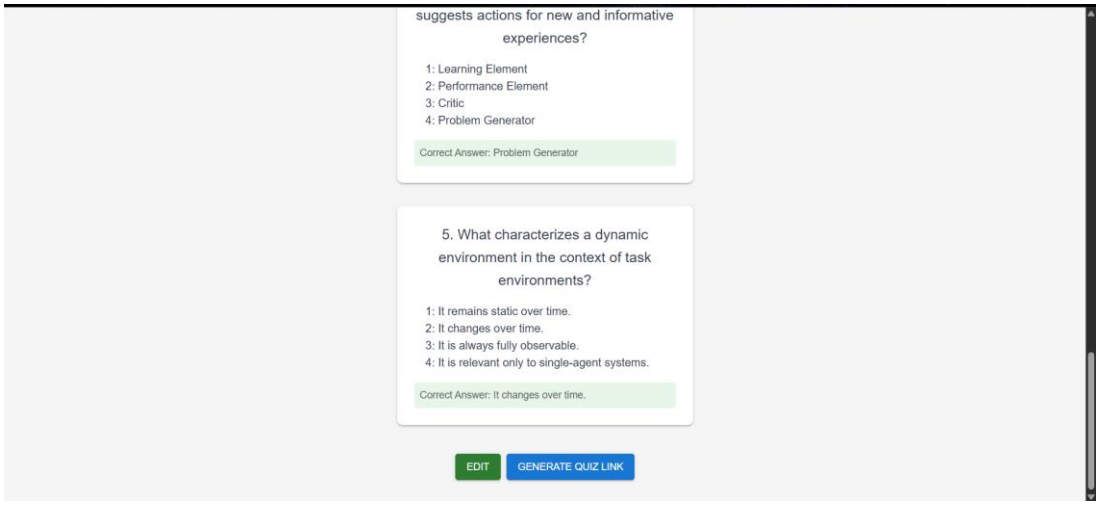
Frontend

Landing Page:

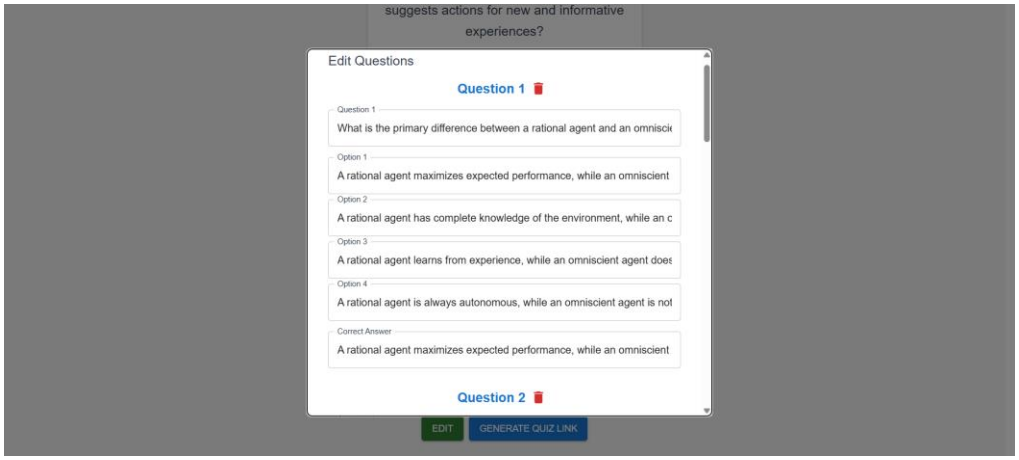


Users can select number of questions (5 or 10)

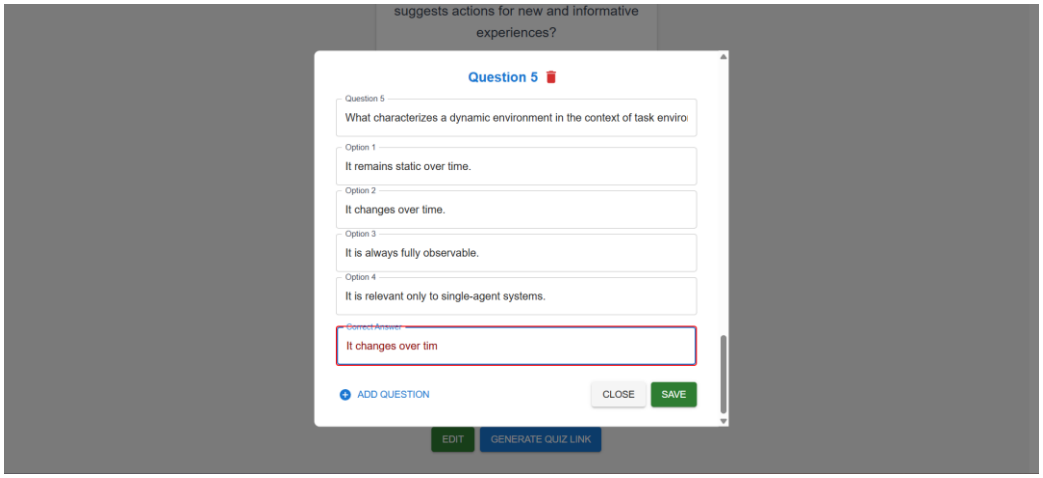




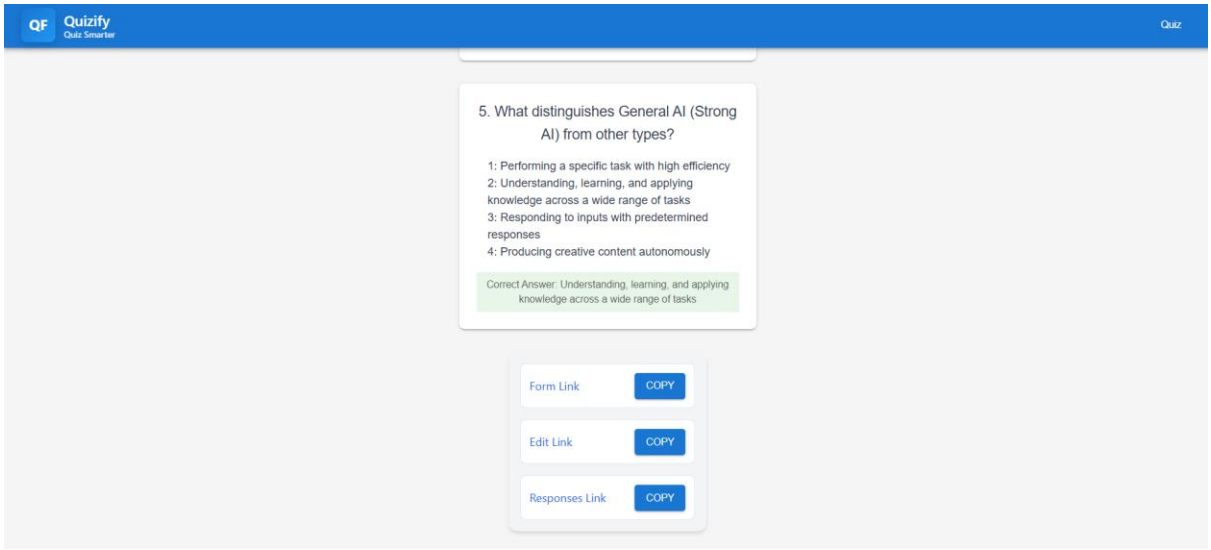
After the questions generates, Users could add, delete or edit questions



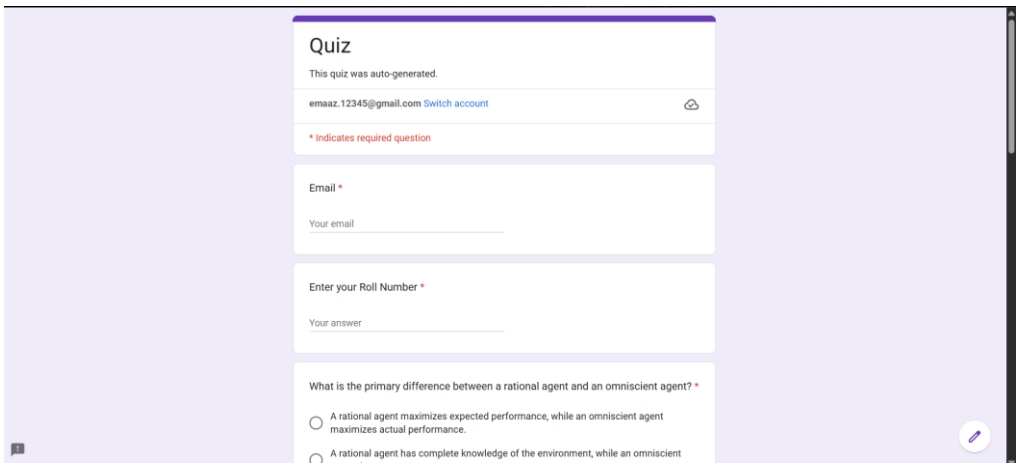
Added validations. Eg: The correct answer must be in the given options.



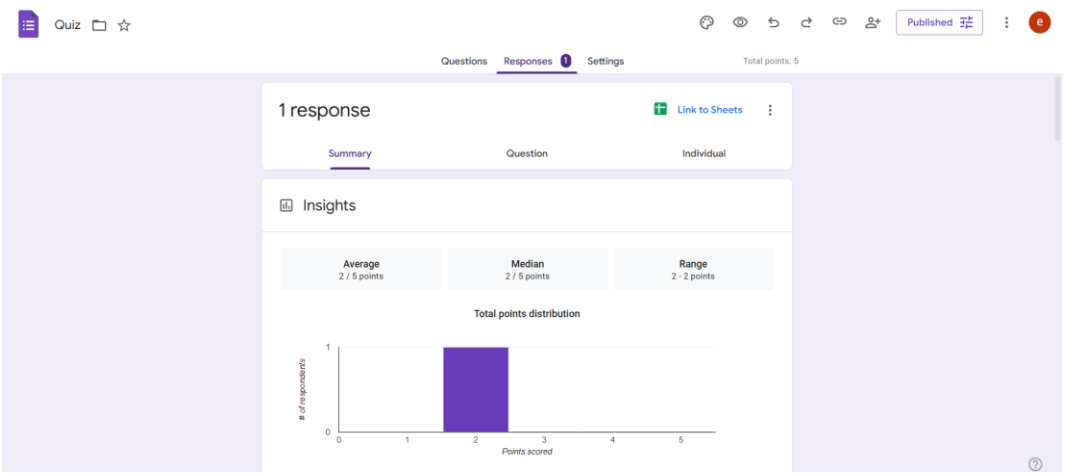
Generating the quiz link gives you three links, Google forms Quiz Link, View responses (spreadSheet) and edit google form link.



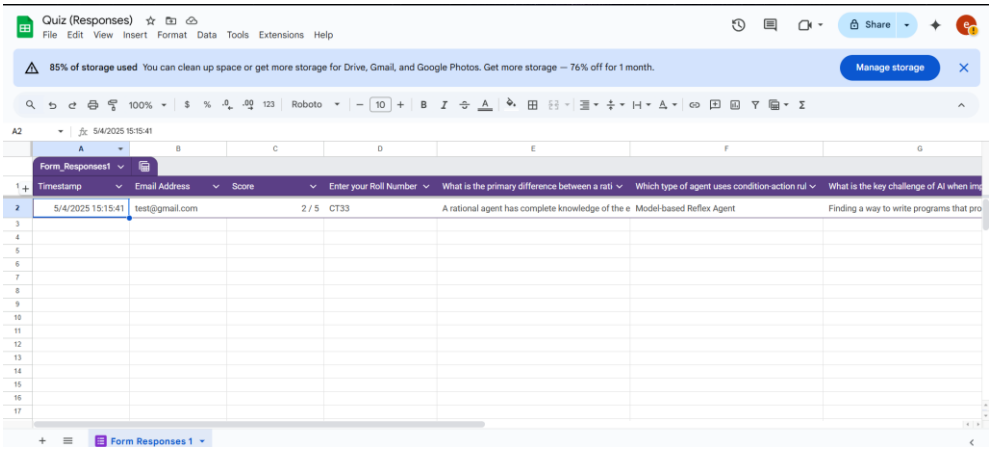
Quiz:



Edit Link:



Reponses Spread Sheet



Future Enhancements

- Support for descriptive/theoretical questions
- User Authentication and quiz history tracking
- Multi-language support
- Integrate question difficulty estimation
- Dashboard for quiz analytics

Conclusion

The Quiz Generation App bridges the gap between passive lecture material and active assessment creation. By combining FastAPI's efficiency with LLaMA 3.3's language understanding and React's UI

capabilities, it empowers educators with a smart, efficient, and customizable tool for quiz generation. The application not only saves time but enhances the overall quality and diversity of quizzes.