The AI-Powered Document Summarizer

Enhancing Textual Comprehension and Productivity with AI

Meet Hitesh Thakkar
Department of Software Engineering
San Jose State University
San Jose, CA, USA
meethitesh.thakkar@sjsu.edu

Aryan Sharma
Department of Software Engineering
San Jose State University
San Jose, CA, USA
aryan.sharma@sjsu.edu

Jainam Chhatbar
Department of Software Engineering
San Jose State University
San Jose, CA, USA
jainam.chhatbar@sjsu.edu

Paavani Karuturi
Department of Software Engineering
San Jose State University
San Jose, CA, USA
paavani.karuturi@sjsu.edu

***Abstract*—This report presents QuizGen, a serverless web application designed to facilitate automated quiz generation from uploaded content. Built on modern cloud-native architectures, QuizGen employs AWS services, OpenAI APIs, and a React-based frontend to provide a scalable, efficient, and user-friendly platform. This document details the system architecture, implementation, challenges, and solutions.**

***Index Terms*—Large Language Models (LLMs), Prompt Engineering, Zero-Hallucination Outputs, Schema Validation, Text Summarization, AI-Powered Question Generation, ChatGPT Integration, OpenAI APIs, Fine-Tuning LLMs, Context-Aware Responses, AI Tools in Education.**

I. Introduction

The demand for automated tools in the education sector has grown significantly, especially for applications that enhance productivity. QuizGen was designed as a serverless solution to generate quizzes dynamically from uploaded documents. It integrates file processing, AI-based content analysis, and cloud-hosted interfaces to deliver a seamless user experience.

The system comprises a backend implemented using AWS Lambda, API Gateway, and RDS, and a React-based frontend deployed on S3. This combination ensures scalability, reduced infrastructure management, and cost efficiency.

II. Methodology

A. Architecture Overview

1). Backend:

Developed using Node.js and deployed with Serverless Framework. Key AWS services used include Lambda (for compute), API Gateway (for routing), RDS (PostgreSQL database), and SSM Parameter Store (for secure environment variables).

2). Frontend:

A React-based single-page application (SPA). Hosted on an S3 bucket with public access and CloudFront for optimized delivery.

3). AI Integration:

Identify applicable funding agency here. If none, delete this.

OpenAI APIs are used to parse document content and generate quiz questions dynamically.

4). The frontend communicates with the backend using REST APIs. The system ensures secure interaction through CORS configurations and signed API requests.

B. Implementation

Backend:

The backend includes Lambda functions for:

-User Authentication: JWT-based authentication to secure endpoints. File Upload: Multer middleware handles file uploads, processed in memory to avoid storage overhead.

-Quiz Generation: Uploaded files (PDF or text) are parsed, and OpenAI APIs are invoked to create questions.

-Database Storage: Sequelize ORM facilitates database operations, storing quizzes and their associated questions in an RDS PostgreSQL instance.

Frontend:

-Built using React, the SPA provides a user-friendly interface for uploading files and managing quizzes.

-Routing is managed with React Router to support subpaths.

-Environment-based configurations enable switching between development and production backend URLs.

CI/CD:

GitHub Actions automates the deployment pipeline:

-Backend: Deployed using Serverless Framework.

-Frontend: Built with environment-specific variables and uploaded to the S3 bucket.

Deployment:

-Backend deployed as a serverless application using AWS Lambda for scalability and cost efficiency.

-Frontend React application hosted on AWS S3 with CloudFront for secure and low-latency content delivery.

-Infrastructure managed using Serverless Framework for seamless deployment and configuration.

-Environment variables securely managed via AWS SSM Parameter Store.

-Integrated GitHub Actions for CI/CD to automate the build and deployment process for both backend and frontend applications.

III. Challenges and Solutions

A. Engineered Prompt Design for Zero Hallucination

We described the output format, such as requiring JSON with specific keys like question, options, and correctAnswer. To guide the model further, we included structured examples demonstrating the expected output format. A post-processing validation script was introduced to ensure API responses conformed to the required schema before saving them to the database.

B. CORS Issues

Initial configurations caused blocked requests between frontend and backend due to CORS. Explicit headers and correct origin configuration resolved the issue.

C. Lambda-to-RDS Connectivity

Timeouts occurred due to Lambda being outside the VPC. Moving both Lambda and RDS into the same VPC addressed the connectivity problem.

D. File Upload and Parsing

Switching from disk storage to memory storage for Multer reduced temporary storage overhead but introduced parsing errors. Adjustments in file handling logic resolved this.

E. Subpath Loading in React

The frontend failed to load subpaths hosted in S3. Adding an S3 bucket policy and configuring the index.html fallback resolved routing issues.

IV. Evaluation

Evaluation of the platform on sample PDF documents and user feedback provided the following insights:

Performance: Lambda execution is efficient, with a maximum timeout of 120 seconds. The SPA loads quickly via S3 and supports smooth API interaction.

Scalability: Serverless architecture scales automatically based on demand.Decoupling backend and frontend ensures independent scaling and deployment.

Cost Efficiency: Pay-as-you-go AWS services reduce operational costs.

Security: Sensitive environment variables (e.g., database credentials, API keys) are securely stored in AWS SSM Parameter Store.JWT-based authentication ensures endpoint security, protecting user and quiz data.

Search Functionalities: Serverless architecture operates on a pay-as-you-go model, minimizing costs during periods of low usage.Hosting the frontend on S3 provides an inexpensive yet highly available distribution mechanism.

Usability: The intuitive SPA design simplifies user interaction, enabling educators to quickly upload content and generate quizzes. Responsive design ensures usability across devices.

Robustness: Error handling mechanisms, including validation and fallback logic, enhance reliability. File upload and processing pipelines ensure compatibility with multiple document types (PDF, plain text).

V. Future Work

A. AI Enhancements

-Improve quiz question generation by leveraging advanced NLP models for contextual understanding. -Expand compatibility to include multilingual content for global accessibility.

B. Mobile Application

-Develop a mobile app for iOS and Android to increase user accessibility and engagement.

C. Enhanced User Features

-Allow users to manually edit or curate AI-generated quiz questions. -Add the ability to upload additional file formats, such as Word documents and PowerPoint slides. -Provide real-time previews of generated quizzes before storage.

D. Analytics and Reporting

-Implement user analytics to track quiz performance and user engagement. -Add visual dashboards for educators to review quiz statistics.

E. Integrations

-Integrate with Learning Management Systems (LMS) such as Moodle or Canvas for seamless adoption. -Offer webhook-based API access for third-party developers.

F. Deployment and Optimization

-Utilize AWS CodePipeline for end-to-end CI/CD. -Optimize AI API calls to reduce latency and improve response time.

G. Sustainability

-Implement a cleanup mechanism to delete unused Lambda logs and S3 files. -Transition to AWS Graviton-based Lambda functions for improved energy efficiency.

VI. Conclusion

QuizGen demonstrates the power of serverless architectures in creating scalable, cost-efficient, and user-centric applications. By leveraging cloud services and AI, the platform provides a robust solution for educators, enhancing productivity and automating routine tasks.

- Incorporating multi-document summarization capabilities
- Enhancing the system's ability to handle domain-specific jargon and technical terminology

- Implementing real-time collaboration features for team-based document analysis
- Exploring the integration of computer vision techniques for processing visual elements within documents
- Future iterations will prioritize enhancing personalization, integrating real-time data processing capabilities, and expanding accessibility. Through continuous innovation, the summarizer aims to redefine how information is consumed and utilized in the digital age.

REFERENCES

[1] AWS Lambda Documentation," Amazon Web Services, 2024. [Online]. Available: https://aws.amazon.com/lambda/. Lewis, M., et al. (2020). "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension." arXiv preprint arXiv:1910.13461.

[2] Serverless Framework," Serverless, 2024. [Online]. Available: https://www.serverless.com/.

[3] "OpenAI API," OpenAI, 2024. [Online]. Available: https://platform.openai.com/.