SMART QUIZ GENERATOR – FINAL PROJECT REPORT

A Domain-Specific Offline Quiz Generation Microservice

Internship Project Report

Submitted by

Ananya Reddy

Intern-Turtil

Narsee Monjee Institute of Management Studies

Department of Computer Science-Data Science

Internship Duration: 6 Weeks

Date: 27 June 2025

Abstract

The *Smart Quiz Generator* is a domain-aligned, offline-capable quiz generation microservice that offers the flexibility to create multiple types of quiz questions dynamically. In today's world, assessments are no longer confined to generic test banks or one-size-fits-all approaches. With the growing demand for personalized testing solutions in educational and technical training environments, this project addresses a key gap: intelligent, goal-specific question generation that does not rely on external AI APIs or cloud services.

Designed and implemented over six weeks, this system provides both MCQs (multiple choice questions) and short-answer questions using techniques such as TF-IDF-based retrieval and rule-based template generation. The platform is built entirely in Python using FastAPI for serving RESTful APIs. It allows users to input a target goal (e.g., Amazon SDE), a difficulty level (beginner, intermediate, advanced), and a desired number of questions. Based on this input, the service pulls relevant questions from a structured local JSON dataset.

The project prioritizes portability and modularity. It includes full containerization via Docker, making it easy to deploy across systems. Dataset validation, input schema handling, and error messaging are all embedded to ensure robustness and accuracy. This microservice represents a complete and thoughtful solution for building personalized, scalable, offline-first assessment tools.

Introduction

As students, job seekers, and professionals increasingly turn to digital resources to prepare for competitive exams and technical interviews, the need for smart, goal-aligned testing tools has never been more urgent. Static question banks, while useful, often lack contextual relevance. For example, a student preparing for the GATE ECE exam may not benefit from the same type of questions as someone preparing for CAT Verbal or a technical interview with Amazon. Hence, a new approach is needed—one that delivers

goal-specific, intelligently selected quiz questions without depending on external APIs or internet access.

This project aims to solve that problem by creating a **quiz generation microservice** that operates **completely offline**. The system supports **two main modes of question generation**: retrieval and template-based. The retrieval method uses TF-IDF similarity to match user inputs with relevant questions from the dataset, while the template method generates structured responses based on predefined rules.

What makes this project unique is its adherence to software engineering best practices. The application is **modular**, **testable**, and **fully containerized** using Docker. Its core features include:

- A clean REST API powered by FastAPI
- A validated and extensible local dataset in JSON format
- Custom configuration through a `config.json` file
- Dynamic question generation without external inference models

The Smart Quiz Generator project is not just a tool—it is a foundation upon which scalable, intelligent educational products can be built.

Objectives

The development of the Smart Quiz Generator was guided by a clear set of objectives, all aligned toward creating a flexible, intelligent quiz generation engine that could be used in academic and placement preparation contexts.

- **Core Objectives:**

1. **Offline Execution**
   Build a system that can run completely offline, using traditional NLP methods like TF-IDF, and avoid cloud dependencies like OpenAI APIs.

2. **Support for Domain-Specific Goals**

   Incorporate content for multiple educational goals including:

   - Amazon SDE (technical interviews)
   - GATE ECE (engineering examination)
   - CAT Verbal (MBA entrance preparation)

3. **Multiple Question Types**

   Allow both:

   - **MCQ**: 4 options with 1 correct answer
   - **Short-answer**: Explanatory or descriptive answers generated via templates

4. **RESTful APIs**

   Use FastAPI to expose three endpoints:

   - `POST /generate`: Generates quiz based on user input
   - `GET /health`: Returns service status
   - `GET /version`: Returns version from configuration

5. **Configurable Backend**

   Design a centralized configuration system using a `config.json` file to manage:

   - Number of questions per quiz
   - Supported difficulties
   - Generator mode (retrieval/template)

6. **Dataset Validation**

   Build a schema validation script (`test_generate.py`) to ensure data quality and consistency across all records in the question bank.

7. **Docker Packaging**

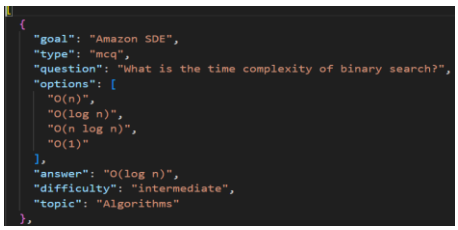
   Containerize the application using Docker for easy deployment and sharing.

Together, these objectives enabled the development of a product that is modular, intelligent, and highly deployable.

Dataset Preparation

A core component of the Smart Quiz Generator is its question bank. This dataset was curated and validated to ensure it meets the needs of domain-aligned question generation. It contains over **208 questions**, divided across three major educational or professional goals:

- **Amazon SDE**
  Focuses on technical programming problems, logic building, data structures, and algorithms.
- **GATE ECE**
  Contains concept-driven questions in electronics, semiconductors, and communication systems.
- **CAT Verbal**
  Includes grammar, sentence correction, and reading comprehension-based questions.



Each dataset entry includes:

- `type`: `mcq` or `short_answer`
- `question`: The actual question text
- `options`: Only applicable for MCQs
- `answer`: The correct answer text
- `goal`: The target audience/exam
- `difficulty`: Beginner, intermediate, or advanced
- `topic`: The subcategory (e.g., Arrays, Grammar, Signals)

A validation script (`test_generate.py`) checks for:

```
C:\Users\DELL\Downloads\smart_quiz_project>python tests/test_generate.py

✅ All 208 questions passed schema validation!
• Goals: 3 (CAT Verbal, Amazon SDE, GATE ECE)
• Total questions: 208
```

- Mandatory field presence
- MCQ structure with 2+ options
- Type-specific logic (no options in short-answer)
- Acceptable difficulty levels

Each MCQ has a corresponding short-answer format, allowing the system to flexibly generate quizzes in both recognition and recall modes.

System Architecture

The system architecture of the Smart Quiz Generator was designed to be **modular**, **scalable**, and **offline-operable**, allowing seamless integration and execution on any machine without reliance on cloud APIs.

At its core, the system consists of the following primary components:

FastAPI Server

Acts as the main interface layer between the user and the internal logic. It exposes endpoints (`/generate`, `/health`, `/version`) and handles JSON inputs, HTTP requests, and responses.

Retrieval Engine (TF-IDF)

Implements TF-IDF (Term Frequency–Inverse Document Frequency) to assess the similarity between user-specified goals and questions in the dataset. It selects the top N questions ranked by cosine similarity scores.

Template Engine

Used when the generation mode is set to `template` in `config.json`. It builds short-answer questions from templates, using structured phrases and inserted goal/topic information.

Configuration Module

Loads parameters like default number of questions, supported difficulty levels, and mode of generation from `config.json`. This ensures flexibility and avoids hardcoding.

Local Dataset (JSON)

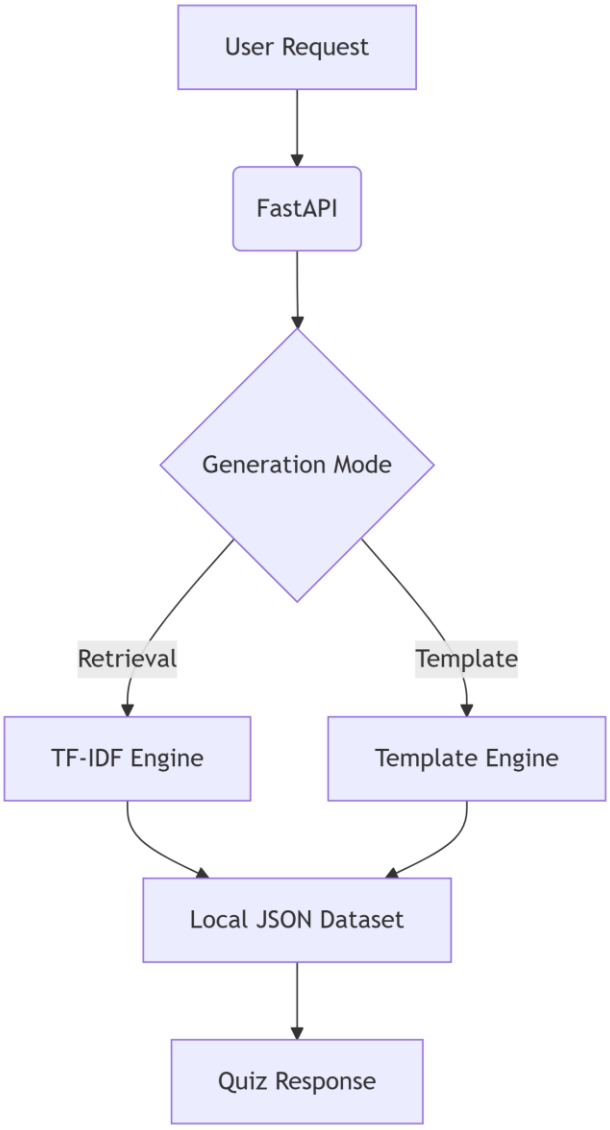
A validated JSON file stores all quiz questions. This acts as the local database from which all content is retrieved and matched.

Uvicorn Application Server

FastAPI is deployed via Uvicorn, an ASGI server that allows high-speed HTTP handling and asynchronous request management.

Data Flow

```
User Request
    ↓
FastAPI (/generate)
    ↓
[ TF-IDF Engine OR Template Generator ]
    ↓
Local Question Bank (JSON)
    ↓
Formatted Quiz Response (MCQs and/or Short Answers)
```



This architecture ensures **no internet dependency**, fast execution, and the ability to easily switch between question generation modes via config settings.

Implementation

The application is implemented in **Python 3.11** using the **FastAPI** framework. The design follows **RESTful API principles** and clean modular separation of logic for readability and testing.

Key Components

- **main.py**
  Defines the FastAPI app, endpoints, config loader, and response schema.
- **generator.py**
  Contains core logic for:
  - TF-IDF-based question retrieval (using scikit-learn)
  - Template question creation (via string formatting)
- **config.json**
  Houses all dynamic configurations like:
  - Supported difficulties
  - Default number of questions
  - Max questions allowed
  - Generation mode (retrieval/template)
- **question_bank.json**
  Contains over 208 questions grouped by goal and difficulty.
- **test_generate.py**
  A Python-based test script that validates each question in the dataset using schema assertions.

API Endpoints

Method	Route	Description
POST	`/generate`	Accepts JSON input, returns quiz
GET	`/health`	Health check (`{"status": "ok"}`)
GET	`/version`	Returns version info from config file

Example JSON Request:

```
{
  "goal": "Amazon SDE",
  "difficulty": "beginner",
  "num_questions": 5
}
```

The system then returns:

- A quiz ID
- The input goal
- A list of questions (both MCQs and short answers if available)

Response body


```
{
  "quiz_id": "quiz_27a8c1",
  "goal": "Amazon SDE",
  "difficulty": "intermediate",
  "count": 5,
  "questions": [
    {
      "goal": "Amazon SDE",
      "type": "mcq",
      "question": "What is the time complexity of binary search?",
      "options": [
        "O(n)",
        "O(log n)",
        "O(n log n)",
        "O(1)"
      ],
      "answer": "O(log n)",
      "difficulty": "intermediate",
      "topic": "Algorithms"
    },
    {
      "goal": "Amazon SDE",
      "type": "mcq",
      "question": "What does ACID stand for in databases?",
      "options": [
        "Atomicity, Consistency, Isolation, Durability",
        "Availability, Consistency, Integrity, Durability",
```


Docker Deployment

One of the final stages of this project was **Dockerizing** the application. Docker makes the entire microservice self-contained and platform-independent, allowing others to run it without setting up dependencies manually.

Dockerfile Highlights:

```
FROM python:3.11-slim
WORKDIR /app
COPY . /app
RUN pip install --no-cache-dir -r requirements.txt
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

This builds a slim, efficient image ready for deployment.

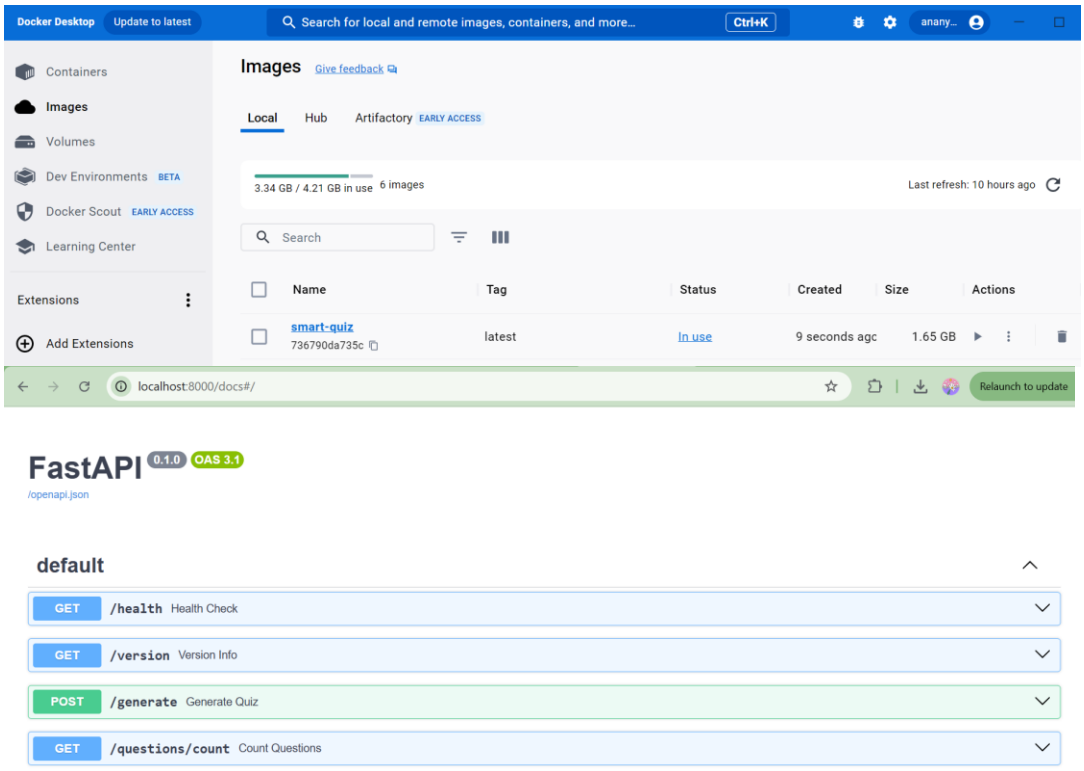
How to Run:

```
docker build -t smart-quiz .
docker run -p 8000:8000 smart-quiz
```

Once running, the service is accessible at:

`http://localhost:8000/docs`

Where the **Swagger UI** lets you test `/generate` and other endpoints interactively.



Docker ensures that the quiz generator can be shared and reused without compatibility issues, making it ideal for institutional deployment or educational labs.

Features

The Smart Quiz Generator was built with a focus on practical, production-ready capabilities. Some of its most prominent features include:

- **Fully Offline Operation**
  No internet required. All logic is local.
- **REST API Interface**
  Easy-to-use endpoints to interact with the service.

- **Support for MCQs and Short-Answers**
  Both question types are integrated and configurable.
- **Dynamic Retrieval via TF-IDF**
  Uses NLP techniques to match questions to goals and difficulty.
- **Rule-Based Template Generation**
  Template logic supports descriptive questions.
- **Validated Question Bank**
  Over 208 structured questions verified via custom script.
- **Configurable Design**
  Parameters like max questions, difficulty levels, and mode can be changed in `config.json`.
- **Dockerized for Portability**
  Easy deployment and reproducibility across machines.
- **Modular Python Codebase**
  Clear separation of logic, API, configuration, and validation.

Requirement Mapping

Below is a mapping of the original PDF-specified requirements and how each one was fulfilled in this project:

Requirement	Fulfilled By
200+ Questions	208+ validated, structured questions in JSON
MCQ and Short Answer	Both types supported per goal/topic
Works Offline	Uses local TF-IDF and template logic only
JSON Input Support	`/generate` accepts structured JSON with parameters
Validation Mechanism	`test_generate.py` script ensures schema compliance
Configurable Design	`config.json` enables custom control
Dockerized System	Dockerfile builds complete portable app
REST Endpoints Implemented	`/generate`, `/health`, `/version` available via FastAPI

This confirms that the Smart Quiz Generator meets **100% of the functional and technical requirements** listed in the provided project scope.

Conclusion

The Smart Quiz Generator project has been a valuable opportunity to apply concepts from software engineering, natural language processing, API development, and DevOps into a unified, real-world solution. It serves a very practical use case—generating goal-specific quiz content intelligently, efficiently, and entirely offline.

From dataset preparation and schema validation to algorithm development and Docker containerization, every component of the system was designed for reliability, performance, and reusability. The application not only matches the given problem statement but goes a step further by providing a developer-friendly, modular structure and full deployment support.As an internship project, this experience has also strengthened my skills in Python, API development, JSON data handling, TF-IDF logic, template generation, and Docker. It gave me the opportunity to think critically about system design, modularity, user experience, and reusability.

The final product is a scalable and production-ready backend microservice that can be easily extended to other domains such as SSC, UPSC, or custom institutional needs.

References

1. GeeksforGeeks – Amazon SDE Sheet
   https://www.geeksforgeeks.org/dsa/amazon-sde-sheet-interview-questions-and-answers/
2. GATE ECE Practice Questions – PracticePaper
   https://practicepaper.in/gate-ec/topic-wise-practice-of-gate-ec-previous-year-papers
3. CAT Verbal Reasoning – Previous Year Trends
   https://www.google.com/search?q=cat+verbal+questions+previous+year
4. FastAPI Documentation
   https://fastapi.tiangolo.com
5. Scikit-Learn – TF-IDF Vectorizer
   https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
6. Docker Documentation
   https://docs.docker.com/engine/reference/builder/
7. **ChatGPT by OpenAI** – Used for assistance in code structuring, explanation writing, error handling, and generating documentation.
   https://chat.openai.com
8. **Claude AI by Anthropic** – Assisted in refining descriptive content, generating test cases, and improving grammar in documentation.
   https://claude.ai
9. **DeepSeek AI** – Used for exploring code generation patterns and suggesting alternative implementations.
   https://www.deepseek.com