

RV College of Engineering®
(Autonomous Institution Affiliated to VTU, Belagavi)
Department of Information Science and Engineering



QUESTIFY : SMART QUESTION GENERATOR

Experiential Learning Report

on

Design Thinking Lab (MIT401L)

Sem: 1

Submitted by

DIVYASHREE H (RVCE24MSE011)

VIBHA V JOSHI (RVCE24MSE014)

M.Tech (Software Engineering)

Submitted to

Prof. Rashmi R

Assistant Professor

Dept. of ISE

2024-25

(ODD Semester)

Title : Questify – Smart Question Bank Generator

CHAPTER 1

Project Overview

Objectives

- **Automate Text Summarization:** The application automatically summarizes key content from uploaded file types such as .pdfs,.docs,.htmls to help students quickly grasp essential concepts, reducing time spent on manual reading.
 - **Generate Relevant Questions:** It creates meaningful and accurate questions that follows Blooms Taxonomy from the summarized text, helping students actively engage with the material and test their understanding.
 - **Implement Secure NLP Solutions:** Transition from OpenAI's LLM to Ollama to ensure enhanced data privacy and control, addressing security concerns regarding sensitive educational data.
 - **Enable Customizable Content Generation:** Incorporate user-driven and system-generated prompts to allow dynamic adjustment of output, enabling users to customize the type, number, and difficulty of generated questions and summaries based on individual needs.
 - **Create a Standalone Executable for User Accessibility:** Develop a user-friendly executable (.exe) file for easy distribution and deployment, eliminating complex setup procedures and ensuring smooth usability across devices.
-

CHAPTER 2

Tools and Technology Used

The technologies used in the Smart Question Bank Generator application are:

1. OpenAI's LLM (Large Language Model-GPT-4o mini) for generating summaries and processing text.
2. Ollama 0.5.10: A tool that could potentially assist in AI-driven NLP tasks such as text summarization and question generation, enhancing content understanding.
3. PDF Handling: Apache PDFBox JDK 21- A Java library for working with text-based PDFs, which is used to extract text.
4. Tesseract OCR JDK 21: An Optical Character Recognition (OCR) engine used to extract text from image-based PDFs.

Backend Technologies:

1. Spring Boot 3.0: A Java-based framework for building and running web applications, used to provide the backend infrastructure for handling requests and integrating the various components.
2. Python Flask 2.2: A micro web framework for Python, possibly used for handling web requests, providing APIs, or integrating additional AI models in the application.

Front-End Technologies (for User Interface):

1. ReactJS 18: (JavaScript frameworks): Use: For building a dynamic and interactive front-end interface where users can upload PDFs and view the generated questions and summaries

CHAPTER 3

METHODOLOGY

The application follows a streamlined, modular design to enhance study and test preparation. The system is designed to be user-friendly and scalable, enabling both students and educators to efficiently prepare for exams and assessments through automated content summarization and question creation. The process begins with users uploading PDFs, which are then processed for content extraction. The extracted content is analysed and summarized, generating concise key points. From these points, relevant questions are automatically generated.

3.1 Design Overview

The application follows a client-server architecture where users upload PDFs via a web interface. Figure 3.1 shows the Spring Boot backend processes these files, extracting content, summarizing it, and generating questions. Data is stored in a database for future use, ensuring a smooth flow from file upload to final output display.

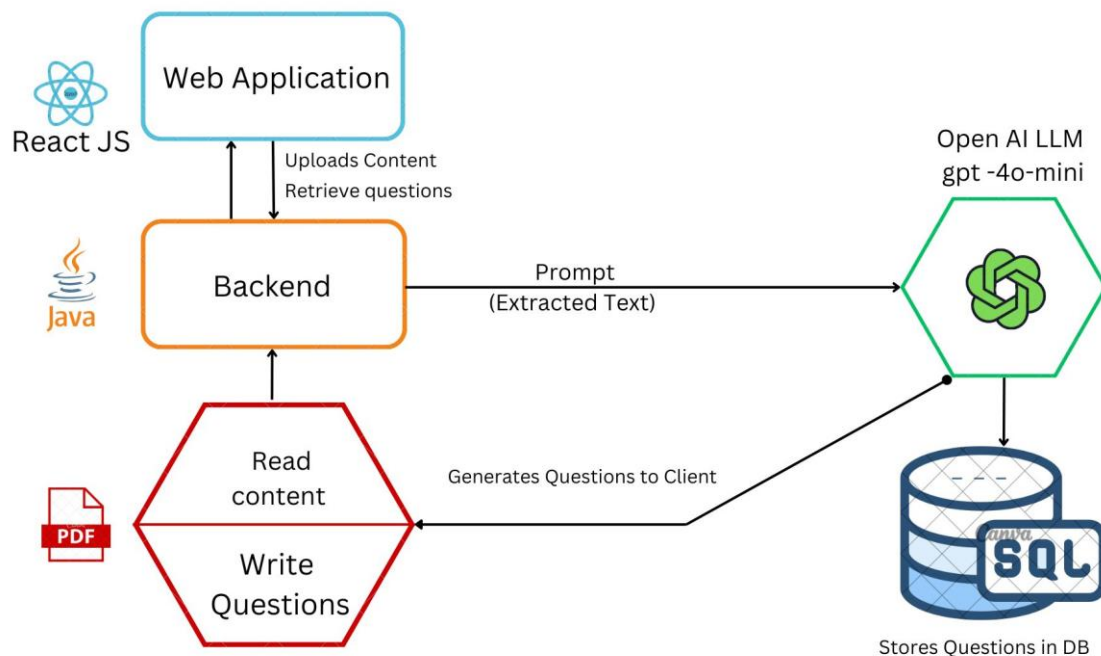


Figure 3.1 Data flow diagram of the proposed system architecture

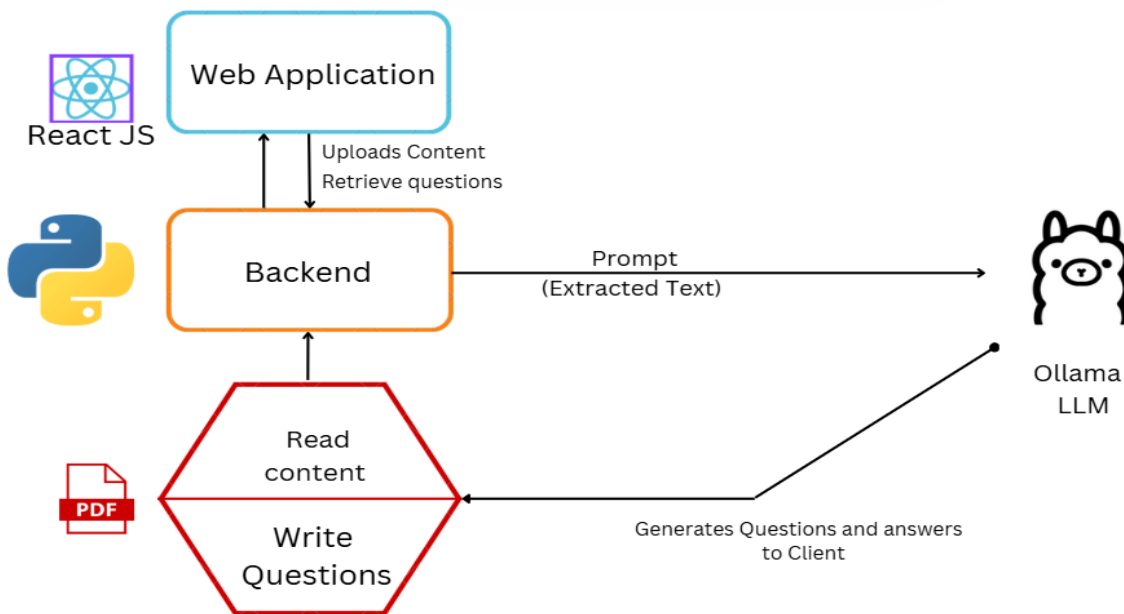


Figure 3.2 Dataflow diagram of the later application version

The application follows a client-server architecture where users upload PDFs via a web interface. Figure 3.2 shows the Python Flask backend processes these files, extracting content, summarizing it using Ollama, and generating questions which is achieved in later versions of the application.

3.2 Use Case Diagram

The Use Case Diagram Figures 3.3 illustrates the interactions between users and the system, depicting key functionalities like uploading PDFs, extracting text, generating questions, and viewing results. It helps in understanding user roles and system boundaries.

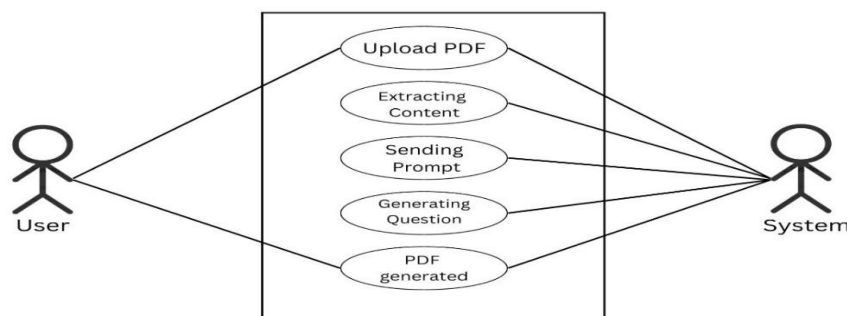


Figure 3.3 Use Case Diagram

CHAPTER 4

Implementation of the Five Phases of Design Thinking

1. Empathize Phase:

- **Feature Identification:** Based on user insights, designed key features such as the ability for users to select the number of questions, difficulty level, and question types based on Bloom's Taxonomy, to generate meaningful and relevant questions for study and assessment preparation.
- **Security as a Priority:** Recognized the importance of data security and ensured that sensitive user data was handled securely throughout the process.

2. Design Phase:

- **Automated Summarization:** Use OpenAI's LLM to automatically generate concise, accurate summaries of uploaded PDFs, reducing time spent on reading.
- **Question Generation:** Create customizable questions based on Bloom's Taxonomy, allowing users to select question type, number, and difficulty to aid active learning.
- **Support for Text and Image-based PDFs:** Use Apache PDFBox for text-based PDFs and Tesseract OCR for image-based PDFs to ensure versatility in document processing.
- **Customizable Question Bank for Educators:** Provide educators with tools to generate tailored question banks, reducing preparation time and improving assessment creation efficiency.
- **User-Friendly Interface & Security:** Develop a simple, intuitive interface for easy interaction while ensuring strong data security for user-uploaded content.

3. Ideate Phase:

- **Initial Solution Exploration:** Initially, we considered using OpenAI's LLM for automated summarization and question generation. The Java Spring Boot framework was chosen for its robustness to manage backend tasks. However, security concerns regarding sensitive educational data arose, prompting us to rethink the architecture.
-

- **Security Enhancement with Ollama:** To address data privacy concerns, we transitioned from OpenAI's LLM to Ollama, ensuring better control over user data while still enabling powerful NLP tasks. Ollama provided a more secure environment for handling sensitive information, enhancing user trust and system integrity.
 - **Switching Backend to Python Flask:** After evaluating different backend options, we decided to move from Java Spring Boot to Python Flask for the backend. Python Flask offers a lighter, more flexible framework, which is easier to work with for integrating AI-driven components and handling web requests. This transition allowed us to improve development speed and focus on integrating the AI models.
 - **User and System Prompts for Customization:** We incorporated user-driven prompts, allowing users to customize the output, such as selecting the type, number, and difficulty of questions. System prompts were also introduced to adjust content generation dynamically based on the context, ensuring that the generated questions and summaries are tailored to the user's needs.
 - **Customizable Question Generation:** A key feature was making the question generation customizable, where users could define the types of questions (e.g., multiple-choice, short answer) and their complexity. This flexibility helped accommodate various learning goals, whether for students looking for practice or educators needing targeted assessments.
 - **Executable (.exe) File Creation:** To enhance accessibility, we developed a standalone executable (.exe) file for easy distribution and deployment. This made the application more user-friendly by eliminating complex setup requirements, ensuring a smooth experience for users across different devices without needing to install additional dependencies.
-

3. **Prototype Phase:** The prototype began with the development of a UI sample using React to ensure all core functionalities were integrated. By focusing on the UI early, we could test key interactions and gather feedback, ensuring the final product would meet user needs before moving forward with backend integration.

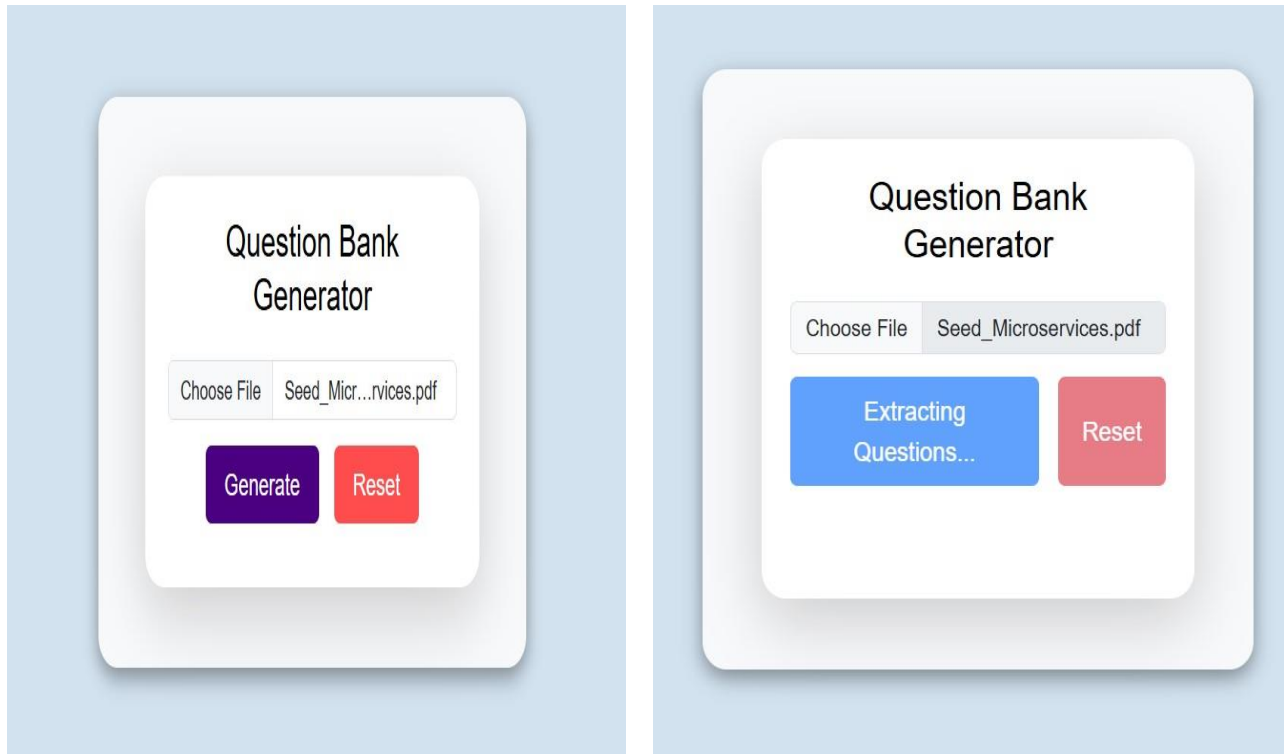


Figure 4.1 Sample UI template designed

The above figures represent initial front-end design allowed us to visualize the user experience and refine features such as PDF upload, text summarization, question generation.

Prototype 1:

In the next phase, the focus shifted to generating questions, with the application producing a fixed set of 10 hardcoded questions using Java Spring Boot with OpenAI's LLM . At this stage, the core functionality was in place to extract content from PDFs and generate basic questions, though without customizable options like the number of questions or difficulty level.

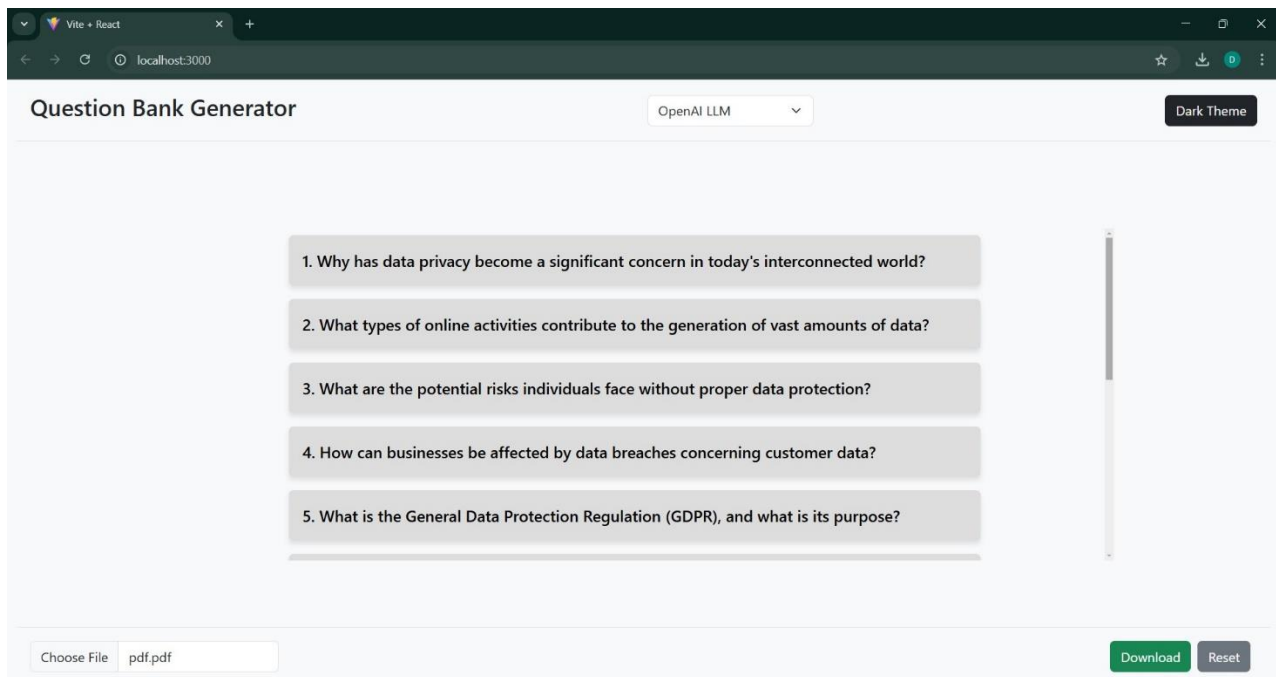


Figure 4.2 Snapshot of Prototype 1 generating fixed number of questions

The UI was enhanced using React and Bootstrap, offering a more refined and user-friendly experience. This phase allowed us to ensure that the core question generation feature worked as expected before adding more advanced features for customization and flexibility in later stages.

Prototype 2:

Advanced Question Generation and File Upload: The next prototype introduced several key features to enhance functionality. Users could now customize question generation based on difficulty levels (basic, intermediate, complex), the number of questions they wanted, and the marks associated with each question (e.g., 2m, 5m, 10m). Additionally, the application expanded its file upload capabilities, allowing users to upload not only PDFs but also HTML and DOCX files, making it more versatile and accommodating to various document types. This phase ensured that the application met more specific user needs, offering greater control over content creation.

The screenshot shows a web browser window with the address bar displaying 'localhost:8080'. The application title is 'Question Bank Generator'. On the right, there is a dropdown menu set to 'OpenAI LLM' and a 'Dark Theme' toggle button. The main interface is divided into a left sidebar and a central content area. The sidebar contains the following controls: a 'Choose File' button and a 'S...df' text input; a 'Select Difficulty Level' dropdown set to 'Intermediate'; a 'Select Marks Alloted' dropdown set to '5M'; a 'No of Questions' input field set to '20'; and 'Generate' and 'Reset' buttons. The central content area displays three generated questions in separate boxes: '16. What are the risks associated with not describing each query and action as a specification in a microservices design? Analyze the potential impact on development.', '17. In the context of the SEED(S) process, how would you assess the effectiveness of a microservice after its implementation? What metrics would you consider?', and '18. Analyze a scenario where actors have been defined too broadly in microservice design. What strategies can be employed to refine these definitions?'. At the bottom of the application, there is another 'OpenAI LLM' dropdown, a green 'Download' button, and a 'Reset' button.

Figure 4.3 Snapshot of Prototype 2 that generates question based on User customised input

Once the core features were integrated, the application was compiled into a JAR file, streamlining the deployment process. This executable format allowed for easy distribution and usage without requiring complex setups.

Generated Questions:

1. Given a scenario where a team is designing a new healthcare microservice, how would you apply the SEED(S) methodology to identify the key actors involved in the service?
2. Analyze a situation where a microservice is designed without proper actor identification. What potential problems could arise from this oversight?
3. How would you apply the interaction pattern discovery step of the SEED(S) process to a microservice that manages patient records? Provide a specific example.
4. In a project where the design of APIs has led to overlapping actor definitions, analyze the implications this could have on service usability and overall system architecture.
5. Describe how you would derive high-level actions and queries for a microservice designed for a retail application. What factors would you consider?
6. Given a case where user feedback on an API specification was largely negative, analyze how this feedback could impact the subsequent steps in the SEED(S) process.
7. How can the principles of product management be applied to improve the actor identification process in microservices design? Provide specific examples.

Prototype 3: Transition to Local Model with Ollama API: After realizing that sending data over OpenAI's LLM posed potential privacy concerns, we decided to build a more secure, local solution.

We integrated the Ollama API to generate questions and answers directly on the user's machine, ensuring that sensitive data would not leave the local environment. The model incorporated all previously established features.

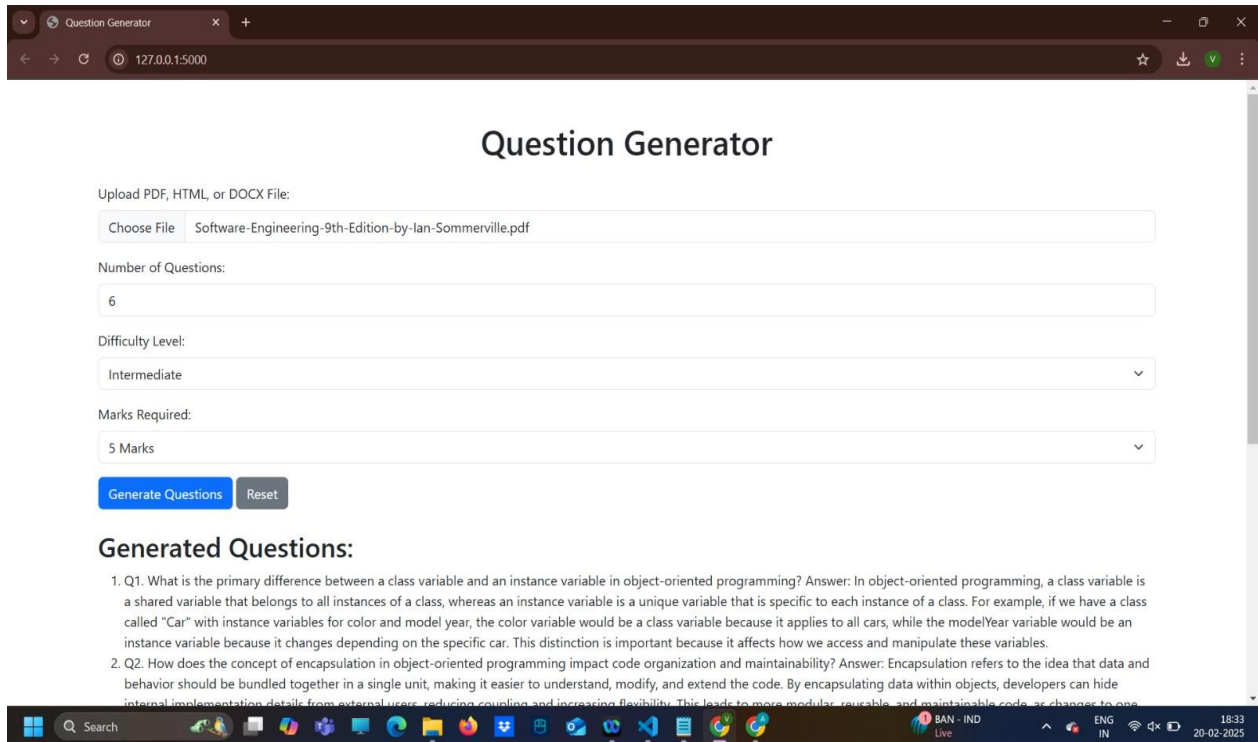


Figure 4.4 Snapshot of Prototype 3 that uses Ollama LLM in a local server that generates answers

Enhanced Flexibility with Python: To improve the flexibility and functionality of the system, we transitioned to using Python for building the application. This shift allowed us to leverage Python's capabilities, providing a broader range of prompt options for question generation. By using Python, we also had greater control over fine-tuning the model and could extend the prompt range, enhancing the precision and customization of the generated content. This local solution, powered by Ollama, provided a more secure and powerful tool for creating tailored study materials.

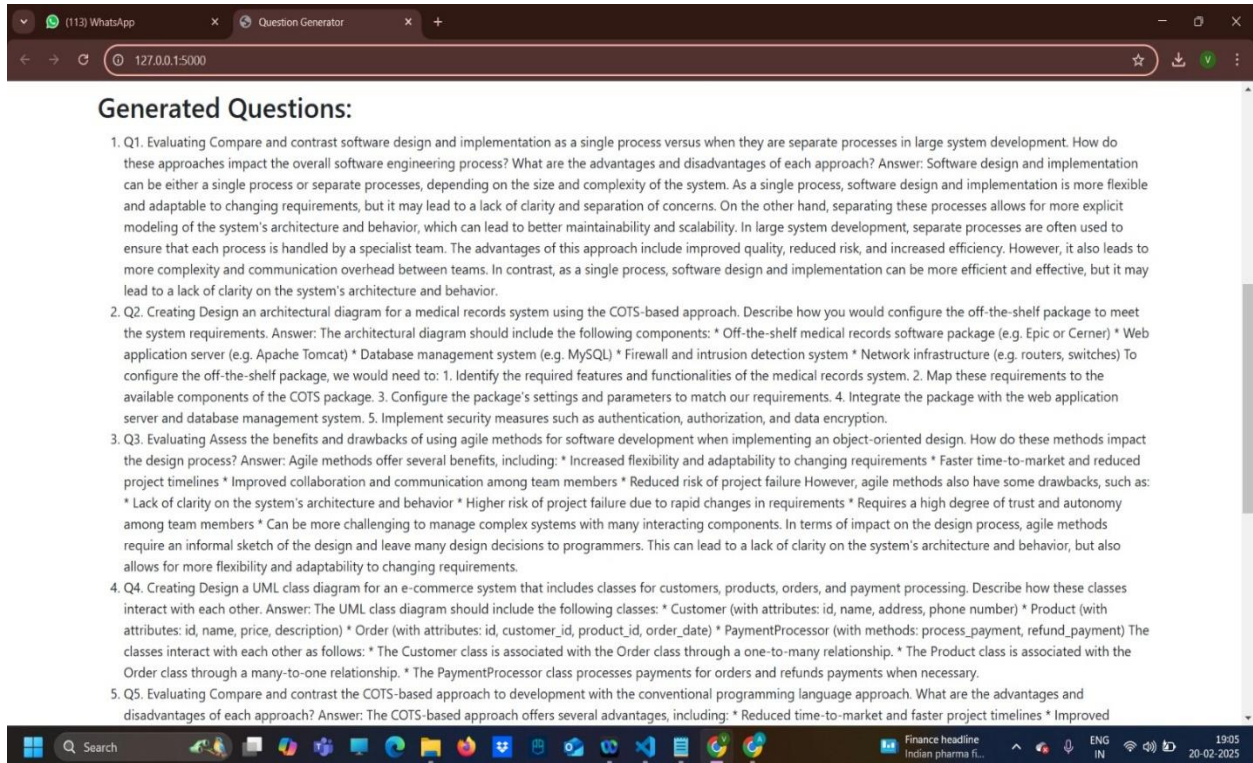


Figure 4.5 Snapshot of generated answers for respective questions

Answer Generation along with Questions: In this phase, we expanded the model to not only generate customized questions but also provide accurate answers for each question. Using the Ollama API, the system now produced both questions and their corresponding answers, ensuring a complete question-answer set that could be used for studying or assessments.

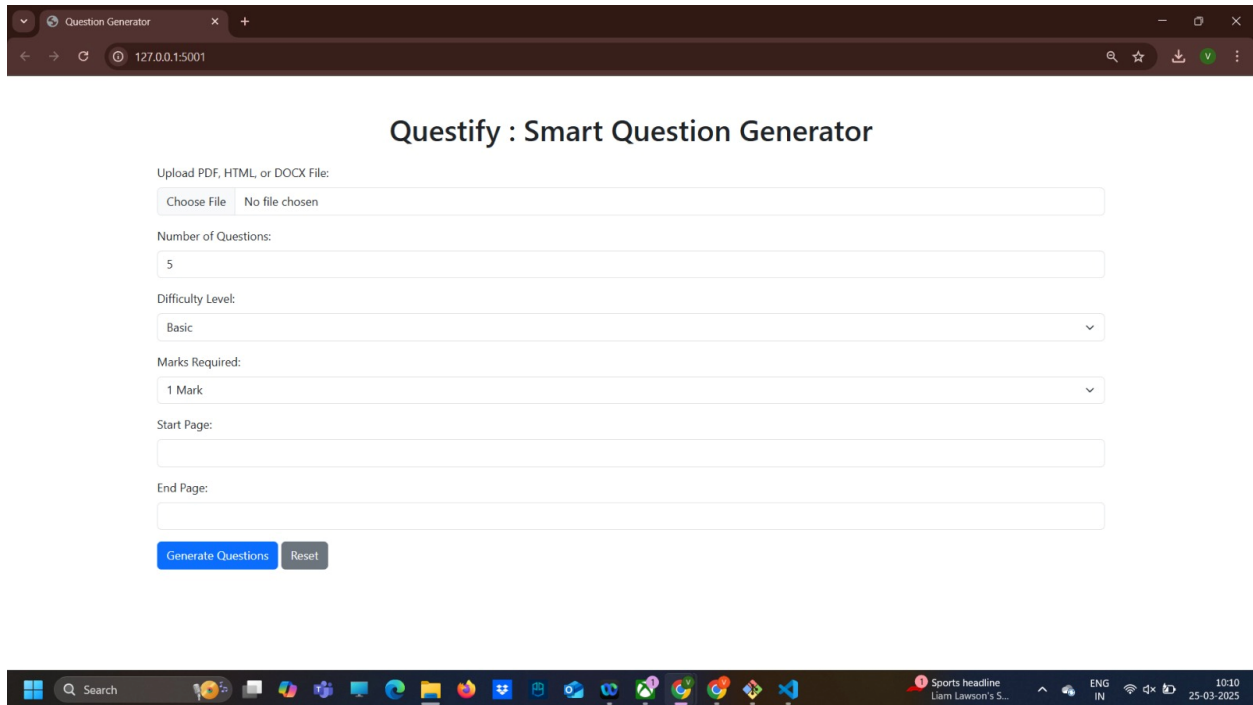
5. Test:

Testing with Large Documents for Stability: The application was thoroughly tested with a large 790-page PDF book, "Software Engineering 9th Edition by Ian Sommerville". Despite the document's substantial size, the application remained stable and successfully processed the content to generate relevant questions and answers. This test demonstrated the system's ability to handle extensive documents while maintaining performance and stability, ensuring that the tool could be used effectively for large textbooks and other substantial educational materials.

CHAPTER 5

Results

Final Prototype:



The screenshot displays the 'Questify : Smart Question Generator' web application in a browser window. The interface includes a title bar with 'Question Generator' and a tab. The address bar shows '127.0.0.1:5001'. The main content area features a title 'Questify : Smart Question Generator' and a form with the following elements:

- 'Upload PDF, HTML, or DOCX File:' section with a 'Choose File' button and 'No file chosen' text.
- 'Number of Questions:' input field with the value '5'.
- 'Difficulty Level:' dropdown menu with 'Basic' selected.
- 'Marks Required:' dropdown menu with '1 Mark' selected.
- 'Start Page:' input field.
- 'End Page:' input field.
- 'Generate Questions' (blue button) and 'Reset' (grey button) buttons at the bottom.

The browser's taskbar at the bottom shows various application icons, a search bar, and system tray information including 'Sports headline', 'ENG IN', and the date '25-03-2025'.

The final version of the Smart Question Bank Generator, built with Flask and Ollama, enhances the prototype by introducing a page number range selection feature. This allows users to specify a particular section of the PDF from which questions should be generated, providing more precise control over content selection. The system extracts text from the selected pages using Apache PDFBox and Tess4J for OCR, ensuring accurate text retrieval. The locally hosted Ollama model then processes the extracted text to generate questions based on user-defined parameters such as difficulty level, marks allotted, and the number of questions. This version ensures efficient and customized question generation, improving usability for educators and students. The frontend, developed with React.js, provides an interactive interface with dropdowns and input fields for easy parameter selection. A reset button enables users to clear inputs and remove uploaded PDFs effortlessly. The system also supports downloading generated questions in DOCX format, making it convenient for study materials and assessments. Overall, this version enhances flexibility, accuracy, and user experience.

Security Enhancement with Ollama API: The transition from using OpenAI's LLM to the Ollama API ensured better control over sensitive user data. By processing data locally on the user's machine, privacy concerns were addressed, creating a more secure environment for data handling. This shift also reinforced the importance of data security in the development of educational tools.

Customization and Advanced Functionality: The application introduced advanced features, such as customizable question generation, where users could select the number of questions, their difficulty, and the type of questions they desired. This customization improved the user experience by allowing the application to cater to various learning goals and assessment needs.

Support for Multiple File Formats: The backend was enhanced to support not only PDFs but also HTML and DOCX file uploads. This increased the versatility of the system, enabling users to process different types of educational content seamlessly.

Answer Generation for Question Sets: The system was expanded to generate both questions and corresponding answers, ensuring that users had complete question-answer sets for study or assessments. This feature provided a more comprehensive solution for educational content creation.

Executable (.exe) for Easy Distribution: To simplify the deployment process, the application was packaged as a standalone executable (.exe) file, removing the need for complex setup procedures. This made the tool more accessible across different devices without requiring additional dependencies.

Inferences

Customization Increases User Engagement: Allowing users to tailor the content generation to their specific needs (e.g., difficulty levels, number of questions) led to a more personalized learning experience. This customization was crucial in catering to different types of users, from students looking for practice to educators needing specific assessments.

Importance of Versatility in File Handling: Supporting multiple file formats (PDF, HTML, DOCX) ensured that the application could accommodate a wider range of educational resources, making it a more versatile tool for various document types and improving its usability.

Streamlined Deployment Enhances Accessibility: Packaging the application as an executable file simplified its distribution and deployment, allowing for a more accessible experience for users without the need for complex installations or dependencies.

Data Security is a Key Priority: The integration of Ollama for local processing emphasized the critical importance of privacy in educational technologies. By ensuring that sensitive data never leaves the local machine, the system reinforced user trust, which is vital for applications handling personal or sensitive information.

Backend Flexibility Improves Efficiency: Transitioning to Python Flask enhanced backend performance by providing a lighter, more flexible framework. This shift allowed for easier integration of AI-driven features and better scalability, ensuring the application could adapt to future needs and growing user demands.

Future Scope

Support for Multi-Language Content: To expand the application's usability, future development could include support for multiple languages. This would allow the tool to cater to a global audience, enabling educators and students worldwide to process study materials and generate educational content in their preferred languages.

Integration with Learning Management Systems (LMS): To enhance the usability of the tool in educational environments, integrating the application with popular learning management systems (e.g., Moodle, Blackboard, Google Classroom) could make it easier for educators to use the generated content within their courses. This would enable seamless distribution of study materials, assessments, and learning activities.

AI-Powered Content Recommendations: Leveraging AI to recommend related study materials, resources, or questions based on a user's learning history could enrich the learning experience. This feature would automatically suggest relevant content to help students focus on areas that need improvement.

Interactive Learning Features: Adding interactive learning features, such as quizzes, flashcards, and instant feedback on answers, could make the tool more engaging. These features could help students actively participate in their learning process, reinforcing the content generated by the application.

CONCLUSION

Successful Development: The project has created an AI-driven tool for automating content summarization and customizable question generation, improving efficiency in study and assessment preparation.

Key Milestones Achieved: Transitioned from OpenAI's LLM to the Ollama API, ensuring secure, local processing of user data. Shifted backend from Java Spring Boot to Python Flask, improving flexibility and development speed. Introduced customizable features for question generation, including difficulty, number, and type of questions.

Significance of Data Security: By focusing on local data processing, the system addresses privacy concerns, ensuring the safe handling of sensitive educational content.

Empowering Users: The tool provides both students and educators with personalized, adaptable content, allowing them to generate study materials and assessments tailored to specific needs.

Impact on Education: The project simplifies the creation and management of study content, improving efficiency and enabling more targeted learning experiences.

Foundation for Future Enhancements: The project lays the groundwork for future developments, such as multi-language support, AI-powered tutoring, and integration with learning management systems.

Overall Significance: The project contributes to the intersection of AI and education, enhancing personalized learning while ensuring data security, with potential for significant future impact.
