AI GENESIS HACKATHON

AI Paper Generator

Team Members Detail

Name	Lablab Username	Discord Username	Builder Type	Preferred Role
Faisal Shahzad	Faisalaiwala	faisalshahzad_24571#0	AI Developer	Team Leader
Malaika Chughtai	Poise_Ravencnql	malaikachughttai	Creative	Team Member
Rimsha Rani	Rimsha Rani	rimsharani	Backend developer	Team Member
Uma Ammara	Uma Ammara	ammara_24804	Backend developer	Team Member
Raqeeba	Raqeeba	raqeeba_26279	Frontend Developer	Team Member
Muhammad Aashan	aashan123	aashan12349276	Documentation	Team Member

Executive Summary

In today's world, organizations and educators struggle to transform large, unstructured learning materials—such as PDFs, slides, and textbooks—into high-quality assessments. Manual question creation is slow, inconsistent, and requires deep subject expertise. Existing tools often handle only a small part of the workflow, forcing users to switch between multiple platforms for content extraction, summarization, question generation, quality checking, and final exam creation.

Our team recognized this gap and developed an **end-to-end assessment automation platform** that processes multiple PDFs, extracts and embeds the content, analyzes topics, and automatically generates high-quality MCQs with difficulty ratings, Bloom-level tagging, and strong grounding in source documents. The system validates each question through automated quality checks, ranks them, and composes full quizzes, assignments, and exams—all delivered in a consistent JSON format ready for deployment or integration.

We built this platform using a modular architecture following agile methodology. The core technologies include **Gemini and Opus models** for generation and summarization, **Qdrant** for vector search, **Python** for orchestration, and optional **Streamlit/React** for the user-facing interface. This design allows scalability, accurate retrieval augmentation, cost control, and seamless multi-PDF ingestion.

In conclusion, our solution provides a **unified, intelligent, and automated assessment generation system**, eliminating the need for multiple tools and drastically reducing the time and effort required to produce high-quality learning materials.

Keywords: Automated Assessment Platform, Multi-PDF Ingestion, MCQ Generation, Qdrant Retrieval, Bloom's Taxonomy, All-in-One Exam Builder

Table of Contents

Chapter 1

Introduction

1. Introduction

This chapter provides essential details about our project, including the problem we aimed to solve, the motivation behind our solution, the objectives we planned to achieve, and the overall scope and limitations of the system. It also outlines the tools, technologies, and methodology used in the development process. The purpose of this chapter is to clearly present the foundation of our project so the reader can understand why it was created, how it addresses real-world needs, and how its design supports modern educational workflows.

This chapter plays a vital role in guiding the reader through the project's purpose, vision, and technological direction.

1.1 Problem Statement

Today in this world, mostly people are not fit and they have very less time to focus on their fitness. If they use fitness apps for fitness then they need multiple fitness apps for complete fitness because mostly fitness apps focus on the specific part like exercise or calorie tracking etc. People also have less time to switch between these apps especially if they need multiple functionalities. Meanwhile studies show most of the people want all in one fitness app.

Our team understands this problem and develops an application which integrates essential fitness tracking features in a very user friendly environment. Users did not need multiple apps for full fitness due to our application.Our application solves the two major problems one is time and second is all important features in our apps for complete fitness.

1.2 Problem Solution

In today's academic and professional learning environments, educators and organizations face a major challenge: **transforming large, unstructured documents—such as PDFs, course notes, research papers, and textbooks—into high-quality assessments**. Creating MCQs, quizzes, assignments, and exams manually is extremely time-consuming and requires subject expertise, attention to detail, and repeated review cycles.

Most available tools solve only one part of the process—some extract text, some summarize content, others attempt question generation—but none provide a **complete, integrated pipeline**. As a result:

1. Educators must switch between multiple platforms for extraction, summarization, question creation, and formatting.
2. Quality and consistency vary due to manual effort.
3. Scaling assessment creation for large courses or multiple documents becomes slow and inefficient.
4. There is no unified system that ensures grounded questions, validated distractors, difficulty tagging, and structured exam output.

Our team recognized this gap and developed a platform that **automates the entire assessment lifecycle**. The system ingests multiple PDFs, extracts meaningful content, embeds it using state-of-the-art models, generates grounded MCQs, validates question quality, assigns difficulty and Bloom-level tags, and finally composes complete quizzes and exams—all within one application.

By providing a unified, intelligent, and automated solution, our project solves two major problems:

1. **Time:** drastically reducing manual effort required to read, interpret, and convert documents into assessments.

2. **Completeness:** offering all essential assessment-generation features in one integrated platform, eliminating the need for multiple apps or manual workflows.

Objectives:

1. To develop an all-in-one automated platform for generating assessments from multiple PDFs and text documents.
2. To extract, preprocess, and semantically chunk large documents for efficient retrieval and analysis.
3. To generate high-quality MCQs with grounded correct answers and validated distractors using advanced LLM models.
4. To assign difficulty levels and Bloom's taxonomy tags to each question for structured learning design.
5. To store embeddings, chunks, metadata, and results securely using Qdrant and local project databases.
6. To allow users to review generated questions and track quality through automated scoring and validation reports.
7. To support end-to-end export of quizzes, assignments, and exams in a standardized JSON format.
8. To provide a scalable, modular system architecture suitable for future expansion into additional assessment types and LMS integrations.

1.3 Scope

The scope of this project outlines the boundaries, stakeholders, constraints, and milestones necessary to meet the defined objectives. It clarifies what the system will provide, what is excluded, and how the development team approached the project's requirements.

Scope Statement

This project delivers a complete, end-to-end platform designed to automate assessment creation from multiple PDFs and textual sources. The system includes document ingestion, text preprocessing, semantic chunking, embedding generation, and retrieval using Qdrant. It supports topic analysis, automatic summarization, and the generation of high-quality MCQs with validated distractors.

A structured assessment builder allows users to generate quizzes, assignments, and exams based on customizable parameters such as difficulty, topic balance, and number of questions. The platform also includes a quality-control pipeline for grounding checks, duplicate detection, and ranking of generated questions.

The system provides secure data management, robust JSON export of all assessments, and an optional frontend for document upload, question review, and final download.

All features operate within a modular, scalable architecture suitable for academic, corporate, and training environments.

Out of Scope Component

1. No manual subject-expert intervention for rewriting or deeply customizing questions.
2. No integration with external LMS systems (e.g., Moodle, Canvas) beyond basic export formats.
3. No real-time collaboration features for multi-user editing.
4. No adaptive learning system (unless added as a future enhancement).
5. No speech-to-text or video content ingestion; only documents (PDF/text) are supported.

Deliverables

The project deliverables include:

- A fully working assessment-automation system with multi-PDF ingestion, MCQ generation, and exam builder.
- A structured JSON output file containing all generated questions, metadata, and assessments.
- Qdrant collection schema and embeddings for all processed documents.
- A demo frontend interface (Streamlit/React) for PDF upload and question review.
- A complete documentation package, including README, architectural diagrams, prompt guidelines, and module descriptions.
- A Jupyter/Colab notebook demonstrating end-to-end execution of the system.

System Boundaries

The system is designed to operate primarily as a local or cloud-hosted application using Python.
It uses:

- **Qdrant** for vector storage and retrieval,
- **Gemini and Opus models** for embeddings and question generation,
- **Local/Cloud storage** for processed documents and JSON exports.

No wearable devices, mobile app components, or biometric data are involved.
There is no standalone mobile version or native OS dependency.
The system does not function as a web-scale search engine; it is bounded to the documents uploaded by the user.

Stakeholders

1. **Primary Stakeholders:**
   Educators, instructors, trainers, academic institutions, and content creators who need automated assessment generation.
2. **Secondary Stakeholders:**
   The development team, product lead, UI/UX designer, quality-assurance reviewers, and potential integration partners.

Constraints

1. **Time:** Project must be completed within the allotted hackathon/development period.
2. **Tools:** Python, Qdrant, Gemini/Opus APIs, Streamlit/React for UI.

3. **Team:** The project is developed by a six-member team, each assigned specific modules.
4. **Budget:** Cloud API costs must be minimized; no paid proprietary tools are used beyond necessary model calls.

1.4 System Components

System component means that all the modules that we are using in our app. We provide the detail of each module:-

1.4.1 Module 1: Ingestion & Preprocessing

Purpose: Accept PDFs (or text), extract and clean content, split into chunks.

Responsibilities:

- Multi-file upload (UI/API)
- Text extraction with OCR fallback
- Language detection
- Text cleaning (headers, footers, whitespace)
- Chunking by semantic boundaries & token limits

1.4.2 Module 2: Semantic Chunking & Indexing

Purpose: Create searchable chunks for retrieval and analysis.

Responsibilities:

- Deduplicate sentences/paragraphs
- Generate embeddings
- Upsert into Qdrant
- Maintain chunk ↔ source mapping

1.4.3 Module 3: Topic Analyzer / Summarizer

Purpose: Identify topics, summarize sections, produce learning objectives.
Responsibilities:

- Extract per-doc and global topics
- Generate hierarchical topic tree
- Summarize each topic
- Highlight high-density/difficult sections

1.4.4 Module 4: Question Generation (MCQ Generator)

Purpose: Generate conceptual MCQs with correct answers and plausible distractors.

Responsibilities:

- Select QA-worthy sentences
- Apply prompt templates
- Generate correct answer + 3–4 distractors
- Validate answer grounding with source chunks

1.4.5 Module 5: Difficulty & Bloom Mapping

Purpose: Tag questions with difficulty level and Bloom's taxonomy.

Responsibilities:

- Define difficulty rubric
- Classify cognitive skill (remember → create)
- Generate hints & time estimates

1.4.6 Module 6: Quiz / Assignment / Exam Builder

Purpose: Assemble structured assessments with configurable constraints.
Responsibilities:

- Parameterized builder (number of questions, topics, difficulty)
- Template support for quizzes, assignments, exams
- Export JSON, CSV, Moodle XML

1.4.7 Module 7: Quality Control & Ranking

Purpose: Filter, rank, and validate generated questions.
Responsibilities:

- Automated grounding checks
- LLM-based quality scoring
- Duplicate detection & fail-reason tagging
- Human-in-the-loop review queue

1.4.8 Module 8: Distractor Validator & Plausibility Engine

Purpose: Ensure distractors are plausible yet incorrect.
Responsibilities:

- Semantic distance checks
- Factual contradiction checks via retrieval
- Grammar & format consistency

1.4.9 Module 9: Search & Retrieval API

Purpose: Provide fast vector search for retrieval and UI.
Responsibilities:

- Top-K embedding search
- Filter by doc or topic
- Semantic search endpoints

1.5.10 Module 10: Frontend & UX

Purpose: UI for uploads, review, configuration, and export.
Responsibilities:

- Multi-file upload & progress display
- Topic explorer & question review queue
- Assessment builder interface

1.4.11 Module 11: Testing, Metrics & Evaluation

Purpose: Measure question relevance, difficulty, distractor quality, and human acceptance.
Responsibilities:

- Unit tests, end-to-end tests
- Grounding metrics, human annotation sampling
- A/B tests for prompt variants

1.4.12 Module 12: Presentation & Documentation

Purpose: Deliver hackathon-ready slides, video plan, and developer docs.
Responsibilities:

- Slide deck (10–12 slides), 3-min demo script
- README with quickstart & JSON reproduction
- API docs, deployment guide, Colab/Jupyter notebook

1.5 Related System Analysis/Literature Review

In literature review, we explain the weaknesses of popular tools related to automated question generation and assessment creation.

It will help you understand other tools' limitations and the solutions that we implemented in our project.

Here is the detail:

Table 1- 1 Related System Analysis with proposed project solution

Application Name	Weakness	Proposed Project Solution
Quizlet	Allows users to create questions manually or use shared sets, but does not automatically generate MCQs from documents.	Our system automatically ingests PDFs, extracts content, and generates MCQs with correct answers and distractors, saving manual effort..
Kahoot	Focused on quizzes, but questions need to be created manually and do not include contextual grounding or source references.	Our system generates questions with grounded references to source documents, ensuring accuracy and traceability.
Google Forms / Microsoft Forms	Allows quiz creation but lacks automatic content extraction or intelligent distractor generation.	Our system extracts topics, generates questions with plausible distractors, and tags difficulty and Bloom's level automatically.
Socrative	Supports quizzes but requires manual question entry and cannot analyze PDFs or textbooks directly.	Our system accepts multiple PDFs, performs semantic chunking, and generates questions automatically from text content.
TopHat / ExamSoft	Designed for classroom exams, mostly manual setup, no automated content generation from documents.	Our system automates end-to-end pipeline: ingestion → topic analysis → MCQ generation → validation → quiz/exam JSON export.

1.6 Vision Statement

The vision of this project is to create an automated, end-to-end system that generates high-quality MCQs, quizzes, and exams directly from PDF documents. The system aims to reduce manual effort, ensure accurate and grounded questions, support difficulty and Bloom's taxonomy tagging, and provide structured outputs for easy integration with learning platforms, making assessment creation faster, smarter, and more scalable.

1.7 System Limitations

While the system automates the generation of MCQs and assessments from PDFs, it has certain limitations. The quality of generated questions depends on the clarity and structure of the source documents; poorly scanned or unstructured PDFs may lead to lower accuracy. The system relies on LLMs for question generation and distractor creation, which can occasionally produce ambiguous or less relevant items. Although automated validation and grounding checks reduce errors, human review may still be necessary for high-stakes assessments. Additionally, processing very large documents or batches may require significant computational resources and API usage, which could affect cost and response time. Despite these limitations, the system provides a substantial improvement over fully manual assessment creation.

1.8 Tools and Technologies

In this section, we list all the tools, technologies, and third-party integrations used in the development of our automated assessment generation system. These tools enable document ingestion, processing, semantic understanding, MCQ generation, validation, and structured output creation.

Table 1- 2 Tools and Technologies

Category	Tool /Technology	Purpose / Use	Version / Notes
Programming Language	Python	Core backend logic, data processing, orchestration	3.10+
PDF Processing	PyMuPDF, pdfminer.six	Extract text from PDFs	PyMuPDF 1.22.0, pdfminer.six 20221105
OCR	Tesseract OCR	Fallback for scanned PDFs	5.3.0
Embeddings / LLMs	Gemini, Opus	Generate embeddings, MCQs, summaries	API-based (latest)
Vector Database	Qdrant	Store and search embeddings for retrieval	1.2.1
Web Framework /API	FastAPI	REST API for retrieval and frontend integration	0.101.1
Frontend / UI	React, Streamlit, Tailwind CSS	Upload, review, configure assessments	React 18+, Streamlit 1.25, Tailwind 3.3+
Orchestration / Queue	Celery + Redis	Manage parallel tasks, batching, retries	Celery 5.3, Redis 7.0+
Data Analysis / ML	NumPy, Pandas, scikit-learn	Data handling, clustering, validation	NumPy 1.26, Pandas 2.1, scikit-learn 1.3
Testing / QA	Pytest	Unit and end-to-end testing	7.4+

Visualization / Reports	Matplotlib, Plotly	Metrics and evaluation dashboards	Matplotlib 3.8, Plotly 5.16+
Deployment / Containerization	Docker, Kubernetes	Containerization and scalable deployment	Docker 24+, Kubernetes 1.29+
Version Control / Collaboration	Git, GitHub	Source code management, team collaboration	Latest stable

1.9 Project Deliverables

The Automated Assessment Generation project will produce several key deliverables that support its successful development. These deliverables cover various stages of the project life cycle, from planning and design to implementation and final presentation.

1.Project planning document

The first deliverable is the Project Planning Document, which outlines the initial phase of the project. This document includes essential details such as the project title, defined objectives and scope, key milestones, team assignments, and planned tools and technologies for development. It serves as the foundation for the project and guides all subsequent activities

2.Software Requirements Specification (SRS)

The second major deliverable is the **Software Requirements Specification (SRS)**. This document details both functional and non-functional requirements.

- **Functional requirements** include modules for multi-PDF ingestion, OCR fallback, semantic chunking, topic analysis, MCQ generation, distractor validation, difficulty and Bloom taxonomy tagging, assessment builder, quality control, and structured JSON export.

- **Non-functional requirements** include ensuring system performance, scalability, security, usability, and a responsive interface suitable for both demo and production environments.

3.Software Design Specification (SDS)

The **Software Design Specification (SDS)** provides a detailed blueprint of the system architecture. It includes:

- Backend architecture for ingestion, embedding, generation, validation, and export.
- Frontend architecture for file upload, review, and assessment configuration.

- Class diagrams, interaction models, and state diagrams for core modules.
- Database schema for Qdrant, metadata storage, and user/session management.
- User interface design layouts for major screens in the web/Streamlit or React frontend.

4. Prototypes

The **prototype deliverable** showcases early working versions of key features. These include:

- PDF ingestion and chunking workflow.
- Topic analysis and summary display.
- MCQ generation for single chunks with distractors.
- Assessment builder demonstrating small quizzes with difficulty and Bloom tags.
  These prototypes help visualize the system's functionality and workflow before full implementation.

5. Final Project (Web Application)

The central deliverable is the **complete system**. This includes:

- Multi-PDF ingestion with OCR fallback.
- Semantic chunking, embedding, and indexing in Qdrant.
- Automatic topic analysis, MCQ generation, and distractor validation.
- Difficulty and Bloom mapping for each question.
- Assessment builder capable of producing structured JSON exports for quizzes, assignments, or exams.
- Frontend for upload, preview, review, and final export.
  The system integrates LLMs (Gemini/Opus), vector database (Qdrant), and orchestration tools (Celery/Redis) for a fully functional, scalable solution.

6. Final Year Project Report

The **Final Project Report** serves as a comprehensive document summarizing the entire project. It includes:

- Project background and objectives
- Literature review / related system analysis
- Detailed system design and architecture
- Module descriptions, screenshots, and workflow diagrams
- Implementation details and testing results
- Discussion of challenges, limitations, and future scope
  This report provides a complete view of the project from planning to final implementation and demonstration.

1.10 Summary

This chapter provided an overview of the AI Paper Generator System project. It introduced the project's core idea of transforming PDF documents into high-quality MCQs, quizzes, and exams with difficulty and Bloom's taxonomy tagging. The chapter outlined the related system analysis, highlighting the limitations of existing tools and how our system addresses them. The vision statement defined the goals of automation, accuracy, scalability, and structured output. System limitations were also discussed,

acknowledging dependency on source document quality, LLM outputs, and computational resources. The tools and technologies section listed all software, frameworks, and third-party integrations used for development. Finally, the chapter detailed the project deliverables, covering planning documents, requirements and design specifications, prototypes, the final system, and the comprehensive project report. Overall, this chapter establishes a clear foundation for the design, implementation, and evaluation of the system in subsequent chapters.

Chapter 2

Requirements Analysis

2 Analysis

In this chapter, we discuss user classes and characteristics, requirement identification, functional requirement, non-functional requirement and external interface requirements.

2.1 User classes and characteristics

User Class	Characteristics
Educators / Teachers	Users who create assessments for students; need automated MCQ generation, difficulty and Bloom's taxonomy tagging, and structured JSON export for quizzes and exams.
Students / Learners	Users who take generated quizzes or practice assessments; need clear questions, answer explanations, and difficulty-level indication.
Content Developers / Trainers	Users who provide study materials or PDF documents; expect accurate extraction, topic summarization, and reliable distractor generation.
Administrators / Evaluators	Users who monitor quality control, validate generated questions, and manage the system workflow; need review dashboards and ranking metrics.

2.2 Functional Requirements

This section describes the functional requirements of the AI Paper Generator System, detailing system and user interactions. Each feature is outlined with its specific functional requirements.

Functional Requirements-1

Identifier	FR-1
Title	Authentication & User Management
Requirement	The user shall be able to sign up, log in, and manage their account. The system shall verify credentials and maintain secure access to PDF uploads, assessment generation, and exported results.
Source	User Requirements, Security Standards

Rationale	Ensures secure access, personalized project usage, and protection of uploaded documents.
Business Rule (if required)	Passwords must be at least 8 characters; role-based access control is applied for educators, students, and administrators.
Dependencies	None
Priority	High

Functional Requirements-2

Identifier	**FR-2**
Title	Ingestion & Preprocessing
Requirement	The system shall allow users to upload single or multiple PDF/text documents. It shall extract content, clean it (remove headers, footers, whitespace), detect language, and split it into semantic and token-based chunks for further processing.
Source	User Requirements, Project Scope
Rationale	Ensures reliable content extraction and standardized chunking for question generation.
Business Rule (if required)	Max file size per upload; supports only PDF and text formats; fallback to OCR for scanned PDFs.
Dependencies	FR-1(Authentication)
Priority	High

Functional Requirements-3

Identifier	**FR-3**
Title	Semantic Chunking & Indexing

Requirement	The system shall deduplicate sentences/paragraphs, generate embeddings for each chunk, and upsert them into Qdrant while maintaining chunk ↔ source mapping.
Source	User Requirements, Project Scope
Rationale	Enables efficient retrieval and semantic search for MCQ generation and topic analysis.
Business Rule (if required)	Each chunk must maintain reference to source doc_id, page, and chunk_id
Dependencies	FR-1(Authentication), FR-2 Ingestion & Preprocessing
Priority	High

Functional Requirements-4

Identifier	**FR-4**
Title	Topic Analyzer / Summarizer
Requirement	The system shall extract per-document and global topics, generate hierarchical topic trees, summarize each topic, and highlight high-density or difficult sections.
Source	User Requirements, Educational Standards
Rationale	Helps in generating relevant MCQs and providing insight into document structure.
Business Rule (if required)	Topic summaries must be concise and grounded in source chunks.
Dependencies	FR-1(Authentication) FR-3 Semantic Chunking & Indexing
Priority	High

Functional Requirements-5

Identifier	FR-5
Title	Question Generation (MCQ Generator)
Requirement	The system shall generate conceptual MCQs with one correct answer and 3–4 plausible distractors for each chunk. It shall validate answers against source chunks.
Source	User Requirements, Project Scope
Rationale	Automates assessment creation and ensures correctness and quality.
Business Rule (if required)	Correct answers must be grounded; distractors must be plausible yet incorrect.
Dependencies	FR-1 (Authentication) FR-4 (Topic Analyzer / Summarizer)
Priority	High

Functional Requirements-6

Identifier	FR-6
Title	Difficulty & Bloom Mapping
Requirement	The system shall assign difficulty levels (easy, medium, hard) and map each question to Bloom's taxonomy.
Source	User Requirements, Educational Standards
Rationale	Ensures balanced assessments suitable for different cognitive levels.
Business Rule (if required)	Mapping is automated but can be overridden by educators.
Dependencies	FR-1 (User Authentication) FR-5 (MCQ Generation)
Priority	Medium

Functional Requirements-7

Identifier	FR-7
Title	Quiz / Assignment / Exam Builder
Requirement	The system shall assemble questions into quizzes, assignments, or exams with configurable parameters (number of questions, topics, difficulty distribution) and export them as JSON, CSV, or LMS-compatible formats.
Source	User Requirements, Project Scope
Rationale	Facilitates automated generation of assessments ready for distribution.
Business Rule (if required)	Export must include metadata, source references, difficulty, and Bloom tags.
Dependencies	FR-1 (Authentication) FR-5,6 (MCQ Generation, Difficulty & Bloom Mapping)
Priority	High

Functional Requirements-8

Identifier	FR-8
Title	Quality Control & Ranking
Requirement	The system shall automatically validate question grounding, check duplicates, score questions using LLM-based metrics, and allow human review via a dashboard..
Source	User Requirements, QA Standards
Rationale	Ensures high-quality and reliable question generation.
Business Rule (if required)	Flagged questions must include fail reasons; human reviewers can approve/reject.
Dependencies	FR-1 (Authentication) FR-5(MCQ Generation)
Priority	High

Functional Requirements-8

Identifier	FR-8
Title	Distractor Validator & Plausibility Engine
Requirement	The system shall validate distractors for plausibility, semantic distance, factual correctness, and grammar/format consistency.
Source	User Requirements, Project Scope
Rationale	Prevents trivial or misleading distractors and ensures assessment quality.
Business Rule (if required)	Distractors must be within defined semantic similarity thresholds.
Dependencies	FR-1 (Authentication) FR-5(MCQ Generation)
Priority	Medium

Functional Requirements-9

Identifier	FR-9
Title	Search & Retrieval API
Requirement	The system shall provide fast vector search to retrieve top-K chunks by embedding, filtered by document or topic, accessible via REST API..
Source	User Requirements, Project Scope
Rationale	Supports retrieval-augmented MCQ generation and frontend search functionality.
Business Rule (if required)	API must respond within defined latency limits.
Dependencies	Semantic Chunking & Indexing
Priority	High

2.3 Requirement Identifying Technique

In requirement identifying technique, we explain the use cases of our application modules with the help of use cases tables and use cases diagrams for each module and also add a full application use case diagram.

Here is the detail:-

2.2.1 Use case Analysis

Use Case #1

Tables for Use case Description for the Use case App Activities

UC Identifier	**UC1**
UC Name	Upload PDF / Text Document
Primary Actor	Educator / Content Developer
Preconditions	User is authenticated; system is online
Post conditions	Document is stored; extraction & preprocessing begins
Trigger	User selects “Upload Document” in UI
Main Flow	The user selects one or multiple PDF/Text files. System validates file format and size. The system uploads files and stores metadata. The system starts text extraction and OCR if needed.
Alternate Flows	Invalid file format → show error. OCR fails → flag file for manual review.

Use Case #2

UC Identifier	**UC2**
UC Name	Text Extraction, Cleaning & Chunking
Primary Actor	System
Preconditions	Document is uploaded
Post conditions	Cleaned and chunked text is ready for semantic indexing
Trigger	Completion of UC-1
Main Flow	The system extracts all text. The system detects language. Cleans headers, footers, whitespace. Divides content into semantic/tokens-based chunks.
Alternate Flows	Extraction fails → mark as “incomplete”.

Use Case #3

UC Name	**UC3**
UC Name	Semantic Chunking & Vector Indexing
Primary Actor	System
Preconditions	Cleaned text chunks exist
Post conditions	Embeddings stored in Qdrant with mapping
Trigger	Completion of preprocessing
Main Flow	1. System deduplicates sentences/paragraphs. 2. Generates embeddings. 3. Upserts chunks into Qdrant. 4. Maps chunk → source document/page.

Use Case #4

UC Name	**UC4**
UC Name	Topic Extraction & Summary Generation
Primary Actor	System
Preconditions	Indexed chunks exist
Post conditions	Topic tree + summaries stored
Trigger	User opens “Topic Analysis”
Main Flow	1. The system identifies topics per document. 2. Generates topic hierarchy. 3. Summarizes topics. 4. Highlights difficult segments.

Use Case #5

UC Identifier	**UC5**
UC Name	Automated MCQ Generation
Primary Actor	Educator / System
Preconditions	Topics and chunks available
Post conditions	MCQs created with answers and distractors

Trigger	User clicks “Generate Questions”
Main Flow	1. User selects topic/chunk count. 2. System identifies QA-worthy content. 3. Generates MCQs with correct answers + 3–4 distractors. 4. Validates grounding.
Alternate Flows	Not enough content → notify user.

Use Case #6

UC Identifier	**UC6**
UC Name	Difficulty Assignment & Bloom’s Taxonomy Mapping
Primary Actor	System
Preconditions	MCQs exist
Post conditions	Each question is tagged with difficulty + Bloom level
Trigger	Completion of MCQ generation
Main Flow	1. The system analyzes question complexity. 2. Assigns difficulty level. 3. Maps cognitive level from Remember → Create.

Use Case #7

UC Identifier	**UC7**
UC Name	Assessment Builder
Primary Actor	Educator
Preconditions	Question bank exists
Post conditions	Structured assessment is created and ready for export
Trigger	User selects “Build Assessment”
Main Flow	1. The user selects a number of questions. 2. User selects difficulty distribution. 3. The user applies filters (topics, Bloom level). 4. System auto-builds assessment.

Use Case #9

UC Identifier	**UC9**

UC Name	Plausibility & Semantic Validation of Distractors
Primary Actor	System
Preconditions	MCQs generated
Post conditions	Distractors are validated
Trigger	Completion of MCQ generation
Main Flow	1. The system checks semantic distance. 2. Ensures incorrectness via factual checks. 3. Verifies grammar and format consistency.

Use Case #10

UC Identifier	**UC10**
UC Name	Semantic Search for Chunks & Questions
Primary Actor	Educator / System
Preconditions	Vector index available
Post conditions	Relevant chunks/questions retrieved
Trigger	User searches topic/chunk
Main Flow	1. User enters search query. 2. System performs top-K vector retrieval. 3. Displays matching chunks/questions.

Use Case #11

UC Identifier	**UC11**
UC Name	Manual Question Review
Primary Actor	Educator / Reviewer
Preconditions	Questions exist in review queue
Post conditions	Questions updated or approved
Trigger	User opens review dashboard
Main Flow	1. User selects question. 2. User edits stem, options, difficulty, Bloom tags. 3. Saves changes.

Use Case #12

UC Identifier	**UC12**
UC Name	Export Quiz/Exam
Primary Actor	Educator
Preconditions	Assessment created
Post conditions	File exported in JSON/CSV/XML
Trigger	User selects “Export”
Main Flow	1. The user selects the export format. 2. The system generates structured exports. 3. Download becomes available.

Use Case #13

UC Identifier	**UC13**
UC Name	Authentication & User Management
Primary Actor	User
Preconditions	System available
Post conditions	User authenticated
Trigger	User selects Login/Signup
Main Flow	1. User enters email/password. 2. System verifies credentials. 3. Grants access.

2.4 Non-Functional Requirements

This section outlines the non-functional requirements of the AI Paper Generator System that support its functional requirements and ensure high system quality. These requirements focus on reliability, performance, usability, security, maintainability, and scalability, enabling a smooth experience for educators, reviewers, and system administrators.

Table: Non-Functional Requirements

Category	**Requirement ID**	**Requirement Description**
Reliability	NFR-01	The system shall maintain a Mean Time Between Failures (MTBF) of at least 800 hours to ensure stable assessment

		processing.
	NFR-02	The system shall automatically recover from service interruptions during document processing or question generation.
	NFR-03	All uploaded documents, extracted text, and generated assessments shall be securely stored in the cloud to prevent data loss.
	NFR-04	The system shall autosave question generation progress every 30 seconds to avoid work loss during failures.
Usability & Performance	NFR-05	The dashboard and project workspace shall load within 3 seconds after user login.
	NFR-06	MCQ generation for a standard document (10–15 pages) shall complete within 10 seconds under normal load.
	NFR-07	Topic extraction and summarization shall complete within 8 seconds per document.
	NFR-08	The system shall support at least **2,000 concurrent users** without noticeable performance degradation.
Security	NFR-09	All user data, documents, and generated questions shall be encrypted both at rest and in transit.
	NFR-10	The system shall require authenticated login using a secure credential method before any document processing.
	NFR-11	Uploaded documents shall not be shared with other users unless explicitly permitted.
	NFR-12	The system shall comply with privacy standards ensuring assessment data is only accessible by the creator
Scalability	NFR-13	The system shall scale horizontally to handle increased CPU/Memory usage as document size grows.
	NFR-14	The vector database (Qdrant) shall support scaling to millions of chunks without performance loss.
Maintainability	NFR-15	All system modules shall follow modular architecture to simplify updates and debugging.
	NFR-16	Documentation for APIs, prompts, and workflows shall be

		updated with every major release.
Availability	NFR-17	The system shall maintain 99% uptime to ensure reliable access for educators and administrators.
	NFR-18	Scheduled maintenance shall be announced at least 24 hours in advance.

2.5 External Interface Requirements

1. User Interface Requirements

The user interface of the MCQ Generation System is designed to be simple, clean, and efficient, allowing users to upload documents and generate questions with minimal steps. The UI follows modern design principles with consistent typography, icons, and spacing to ensure readability.
The interface provides essential screens such as **Upload PDF**, **Document Preview**, **MCQ Generation Panel**, **Question Editor**, and **Export Options**.

Auto-scaling layouts ensure proper display on devices ranging from small laptops to large desktop monitors. Tooltips, progress indicators, and validation messages are integrated to guide users smoothly through each step, such as upload status, chunking progress, and question generation loading. The system also includes an inline MCQ editor, enabling users to review, modify, and approve generated questions before exporting.

2. Software Interfaces

The system supports both **web-based** and **desktop environments**, depending on deployment choices.
Key software interfaces include:

- **PDF Parsing Library**
  Used for reading, extracting text, and chunking PDF content (e.g., PDFBox / PyPDF2 / PDFPlumber).
  OCR Engine (Optional)
  Tesseract OCR is used when the uploaded PDF contains images or scanned pages.
  LLM API Integration
  The system communicates with LLM APIs (e.g., OpenAI API) to generate MCQs, distractors, and explanations.
  Secure API keys are managed through environment variables.
- **Database**
  A cloud or local database stores user history, generated question sets, and metadata (e.g., Firebase, MongoDB, or SQLite depending on version).

- **Export Modules**
  The app integrates libraries to export MCQs into **PDF**, **docx**, and **CSV** formats.
- **Frontend Framework (Optional)**
  If web-based: React/Next.js for a modern UI.
  If desktop: Electron / .NET WinForms / Python Tkinter (based on your implementation).

3. Hardware Interfaces

The application does not require external hardware, as it runs on standard computers.
However:

- For OCR processing, the system uses the device's CPU/GPU for text recognition.
- Large PDFs may require additional RAM depending on file size.
- No external sensors, Bluetooth devices, or embedded systems are involved.

The system relies on the machine's internal hardware capabilities to process documents and communicate with cloud services.

4. Communication Interfaces

The system requires a stable internet connection for:

- LLM API requests (question generation, validation, improvement).
- User authentication (if login system is included).
- Cloud database synchronization for user-generated content.

Communication with external services is handled over **HTTPS** to ensure secure data transmission.
In case of temporary internet loss, the system caches local progress and resumes uploading or generating once the connection is restored.

If offline mode is enabled, basic operations such as:

- PDF upload
- Text extraction
- Local question history viewing

can be performed, while MCQ generation (LLM-based) requires internet access.

2.8 Summary

In this chapter, we presented the complete analysis of the AI paper Generator System through detailed use cases, use case diagrams, and functional descriptions of each module. The use case tables and diagrams illustrate how users interact with the system—from uploading PDFs to generating, reviewing, and exporting MCQs. Each module was analyzed to show its purpose, workflow, and interaction with other components of the system.

Additionally, the chapter outlined all functional and non-functional requirements essential for the system's performance. Functional requirements describe the core operations such as authentication, PDF processing, MCQ generation, editing, and exporting. Non-functional requirements define the overall quality attributes including performance, reliability, usability, and security.

The external interface requirements further explained the software, hardware, communication, and user interface elements necessary for building and running the system. Overall, this chapter provides a complete understanding of what the system must do and the conditions under which it must operate.

Chapter 3

Design and Architecture

3. System Design

Product Perspective

AI Paper Generator is an intelligent, automated assessment-creation system designed to help educators generate high-quality MCQs, summaries, topic outlines, and exam papers from academic documents. It processes uploaded PDFs or text files, analyzes their content using AI, and produces structured assessments such as quizzes, assignments, and exams. The system integrates multiple backend modules—ranging from ingestion, semantic chunking, indexing, MCQ generation, distractor validation, topic extraction, and quality scoring—to ensure each generated question is accurate, meaningful, and aligned with the source material.

AI Paper Generator is developed as a web-based application and backend service. The system uses OCR-based PDF parsing for scanned documents, vector embeddings stored in Qdrant for fast retrieval, and LLM APIs to generate questions, summaries, and metadata. A dedicated frontend allows users to upload files, explore topics, review questions, configure assessments, and export final paper formats (CSV, JSON, LMS-compatible outputs). All components communicate through secure REST APIs, ensuring smooth flow between ingestion, analysis, and paper generation.

There are some design Constraints,

AI Paper Generator is designed to be highly efficient and responsive while handling large academic documents. PDF ingestion and semantic chunking must complete within a reasonable timeframe, even for long files. Embedding generation and vector indexing must produce results quickly to avoid user delays. LLM-based question generation should respond within a few seconds while maintaining accuracy and contextual grounding.

Switching between core modules—such as Topic Explorer, Question Review Queue, and Exam Builder—should feel seamless. Vector searches, question previews, summaries, and topic trees must display instantly with minimal computational overhead. The system must also avoid UI freezing during bulk operations such as batch MCQ generation or evaluation scoring.

Caching, pipeline optimization, and bulk API calls are used to ensure fast performance and scalability. All quality control mechanisms (such as distractor validation and ranking) must produce reliable results without compromising speed. The overall design ensures that the system remains scalable, accurate, and responsive, even under heavy load or large document processing.

3.1 Design Considerations

Assumptions/Dependencies	Description
Web-based Platform Dependency	The AI Paper Generator is designed as a web application and assumes users will access it through modern browsers on desktops or laptops.
LLM / API Dependency	Core modules—such as MCQ generation, summarization, topic analysis, and distractor creation—depend on external Large Language Model APIs

	for processing.
OCR & PDF Parsing Dependency	The system assumes that uploaded documents are in PDF or text format. Scanned PDFs rely on OCR for extraction, and accuracy depends on scan quality.
Vector Database Dependency	Semantic search, chunking, and retrieval depend on Qdrant or another vector database for storing embeddings and enabling efficient top-K search.
Stable Internet Connectivity	Continuous internet access is required for uploading documents, generating embeddings, calling LLM APIs, and exporting assessments.
User Account Persistence	The system assumes that users remain logged in using session tokens until they log out manually.
Document Quality Assumption	The app assumes that uploaded documents contain readable academic content from which topics, summaries, and MCQs can be extracted.

Limitations of the AI Paper Generator:

- **No 100% Accuracy Assurance:**
  Generated MCQs, summaries, and topics depend on AI models and may require manual review for academic precision.
- **No Real-Time Human Reviewer:**
  The system does not include live expert verification or teacher–assisted correction during question generation.
- **Dependence on Document Quality:**
  Poorly scanned PDFs or low-quality text reduce OCR accuracy and can affect the quality of generated questions.
- **No Offline Mode:**
  The system requires an active internet connection for LLM processing, embedding generation, and indexing.
- **Limited Multimedia Input:**
  The system only supports text/PDF documents; it cannot process images, PowerPoints, or handwritten notes directly.
- **Does Not Replace Teaching Expertise:**
  The tool assists instructors but cannot independently judge curriculum design, academic difficulty alignment, or pedagogy.
- **Scalability Dependent on APIs:**
  Performance may be affected by rate limits or response delays from external AI/embedding services.

3.1.3 Risks

Risk	Mitigation Strategy
Internet Connectivity Failure	Most operations such as LLM processing, embedding generation, and indexing require internet access. To reduce disruption, the system performs local caching of uploaded documents and preserves user progress until connection is restored.
High Load or Slow Performance During Large PDF Processing	The system uses background queues, chunk-based processing, and optimized embedding pipelines to ensure smooth performance even for long academic PDFs.
API Rate Limits or LLM Service Downtime	Implement fallback API endpoints, automatic retry logic, and user notifications when external AI services temporarily fail.
Data Privacy and Security Concerns	All uploaded documents and generated questions are stored securely. Data is encrypted in transit and at rest. Strict access controls ensure only authenticated users can access their files.
Inaccurate or Ambiguous Question Generation	A built-in Quality Control & Ranking module flags weak questions and provides automated validation. The system also allows manual review before export.
OCR Errors for Scanned PDFs	Integrated OCR with preprocessing improves text extraction. The system alerts users when low-quality scans may affect accuracy.
Vector Database Failure or Slow Search	Redundant indexing and backup embedding storage ensure recovery. Query optimization ensures stable semantic search performance.

3.2Architectural Design

The AI Paper Generator follows a Multi-Tiered Architecture based on a modular pipeline combined with a service-oriented design. This structure separates the system into independent layers—ensuring scalability, maintainability, and efficient processing of large documents. The architecture is primarily divided into the Presentation Layer, Application / Processing Layer, and Data Layer.

Application Layer (Processing & Business Logic)

The **Application Layer** handles the core intelligence of the system. It manages the document-processing pipeline, executes AI functions, and controls all intermediary logic. This layer coordinates multiple modules:

- **Ingestion & Preprocessing Module** – Extracts text, applies OCR, cleans content.
- **Semantic Chunking & Indexing Module** – Splits content into meaningful chunks and generates embeddings.
- **Topic Analyzer & Summarizer Module** – Identifies topics, produces structured summaries, and generates topic trees.

- **MCQ Generator Module** – Produces conceptual questions, correct answers, and distractors using LLM APIs.
- **Difficulty & Bloom Mapping Module** – Classifies cognitive level and assigns difficulty tags.
- **Quality Control & Ranking Module** – Scores questions, removes duplicates, and validates distractors.
- **Exam Builder Module** – Assembles quizzes, assignments, and exams based on user settings.

This layer acts as the **controller**, receiving requests from the UI, applying business logic workflows, coordinating with external APIs (LLMs, embeddings), updating the data layer, and sending results back to the UI.

Data Layer (Storage & Retrieval Layer)

The **Data Layer** manages all data storage, organization, and retrieval processes. It includes:

- **Document Storage** – Stores uploaded PDFs and extracted text for reuse.
- **Vector Database (Qdrant)** – Stores embeddings and supports fast semantic search for retrieval-augmented generation.
- **Metadata Repository** – Stores chunk mappings, topic trees, question quality scores, and user configurations.
- **Cache Storage** – Caches repeated API outputs (such as embeddings and summaries) to speed up performance.
- **LLM & Embedding APIs** – External AI services used for processing text, generating questions, and producing summaries.

This layer ensures accurate data persistence, efficient vector retrieval, and scalable storage for large documents and generated assessments.

3.3 Data Design

The AI Paper Generator focuses on how academic documents, generated content, and user data are structured, stored, and processed to ensure consistency, scalability, and fast retrieval. The system uses a NoSQL database (Firebase Firestore) for storing user profiles, preferences, and document metadata, and a vector database (Qdrant) to store embeddings for semantic search. Offline caching is handled using SQLite to improve responsiveness for repeated operations.

Data design ensures that each module—document ingestion, topic analysis, MCQ generation, and assessment building—can access required information efficiently, while preserving the provenance of generated questions and summaries.

3.3.1 Data Dictionary

Table: Alphabetical List of Major Data Entitie

Entity / Attribute	Data Type	Description
Answer	String	Correct answer for a generated question.
Chunk_ID	String	Unique ID for each semantic chunk of a document.
Choices	JSON Array	Stores multiple-choice options for each question.
Document_Name	String	Name of the uploaded PDF or document.
Document_ID	String	Unique identifier for each uploaded document.
Embedding_Vector	Array[Float]	Dense embedding for semantic search and retrieval.
Generated_Question	String	Text of the generated question.
Generated_By	JSON Object	Stores model information (e.g., LLM, prompt template).
Metadata	JSON Object	Stores document metadata like author, title, page count.
Topic	String	Detected topic associated with a chunk or question.
Difficulty	String	Difficulty level of a question (easy, medium, hard).
Bloom_Level	String	Cognitive skill according to Bloom's taxonomy (remember, apply, analyze, etc.).
Confidence_Score	Float	AI-assigned confidence for generated question correctness.

Source_Chunks	JSON Array	References to original document chunks used to generate the question.
User_ID	String	Unique identifier for each registered user.
Assessment_ID	String	Identifier for quizzes, exams, or assignments.
Created_At	Timestamp	Date and time when the question or assessment was generated.
Validated	Boolean	Indicates if the question passed quality control and distractor validation.
Review_Notes	String	Optional notes for human reviewer feedback.

3.4 Design Decisions

Software Design Paradigm: Object-Oriented Design (OOD)

The AI Paper Generator is developed using the object-oriented design (OOD) paradigm, implemented primarily with Java and Kotlin in Android Studio. This approach was chosen because the system involves multiple real-world entities such as documents, chunks, questions, assessments, and users. By representing each entity as an object with its attributes and functions, the system becomes structured, maintainable, and easier to extend. OOD principles like encapsulation, inheritance, and modularity improve code readability, reusability, and scalability.

Architectural Pattern: MVC (Model-View-Controller)

The application follows the Model-View-Controller (MVC) architecture to maintain a clear separation of concerns. The Model layer manages core business logic, including document ingestion, topic analysis, question generation, and embedding management. The View layer handles the interface using Activities, Fragments, and UI components, allowing users to upload documents, view generated papers, and interact with assessments. The Controller layer acts as an intermediary, processing user inputs, orchestrating data flow, and communicating with databases and AI modules. This separation ensures modularity and easier maintenance.

Database Design: Cloud Storage

The AI Paper Generator uses a cloud-based storage system to manage all user data and generated content. This includes user profiles, uploaded documents, generated papers, MCQs, and assessment data. By leveraging cloud storage, the application ensures that all data is securely saved, synchronized across devices, and accessible from anywhere with an internet connection. This approach simplifies data management, eliminates the need for local databases, and supports scalable storage as the number of users and documents grows.

UI/UX Design: Modular and Consistent Design System

The interface is built using a modular design approach. Key components include card views for document management and assessment summaries, progress indicators for generation status, and interactive elements for editing questions. The consistent design system ensures a user-friendly experience, with intuitive navigation and clear visualization of AI-generated outputs.

3.5 Summary

This chapter provided a detailed overview of the design and architectural considerations for the AI Paper Generator application. It covered the class structure, system components, and user interface design, ensuring alignment with the project's objectives. The MVC architecture allows efficient handling of AI-powered features while maintaining scalability and modularity.

Data management is optimized through a combination of SQLite, Firebase Firestore, and Qdrant, ensuring offline support, cloud synchronization, and fast semantic search. Object-Oriented Design principles support maintainable and extendable code, while the UI/UX design ensures a simple, intuitive, and modular interface.

Design decisions in this chapter collectively ensure that users can generate papers, extract topics, create MCQs, and manage assessments efficiently, with automated AI processing and a responsive, easy-to-use interface.

Chapter 4
Implementation

4. Implementation

This chapter discusses implementation details of the project with the help of pseudocode algorithm:-

4.1 Algorithm

Tables of Algorithms

Splash Screen
Input: None
Output: Redirect to Home or Login

Start
  if isUserLoggedIn() == true then
  navigateTo("HomeScreen")
  else
  navigateTo("LoginScreen")
  end if
End

User Login Algorithm
Input: Email, Password
Output: Successful login → Home Screen

Start
  get email, password
  if email OR password is empty then
  showMessage("Fields cannot be empty")
  else
  if authenticateUser(email, password) == true then
  navigateTo("HomeScreen")
  else
  showMessage("Invalid Login Credentials")
  end if
  end if

End

User Signup Algorithm

Input: Name, Email, Password

Output: Account Created → Login Screen

```
Start
   get name, email, password

   if anyFieldEmpty() == true then
      showMessage("All fields are required")
   else
      if isEmailAlreadyRegistered(email) == true then
         showMessage("Email already exists")
      else
         createNewUser(name, email, password)
         showMessage("Account created successfully")
         navigateTo("LoginScreen")
      end if
   end if
End
```

Home / Dashboard Loading Algorithm

Input: User ID

Output: Display Dashboard Options

```
Start
   loadUserPreferences(UserID)
   displayOptions(["Generate Paper", "Generate MCQs", "Summaries", "Chatbot"])
End
```

Upload File Algorithm (PDF / DOC / TEXT)

Input: File

Output: Extracted Text

```
Start
    mealList ← fetchMealsFromDatabase(selectedMealGoal)

   for each meal in mealList do
      displayMealName(meal.name)
      displayMealImage(meal.image)
      displayMealInfo(meal.calories, meal.protein, meal.description)
   end for

   if userAddFavorite(item) == true
     addFavorite(item)
End
```

Track Calories

Input: intakeCalories, burnedActivity (type, duration), intakeByFood, SetGoal, Undo Entry

Output: Progress bar update, goal notification, SQLite storage

```
Start
   openFileChooser()

   if fileSelected == false then
      showMessage("No file selected")
   else
      if fileFormat == PDF then
         extractedText = extractPDF(file)
      else if fileFormat == DOC/DOCX then
         extractedText = extractDOC(file)
      else if fileFormat == TXT then
         extractedText = readTextFile(file)
```

```
    else
        showMessage("Unsupported file type")
    end if

    storeText(extractedText)
    navigateTo("TextPreviewScreen")
  end if
End
```

AI Paper Generation Algorithm
Input: Topic or Uploaded Text
Output: AI-Generated Paper

```
Start
  get inputText or topic

  if inputText is empty then
    showMessage("No input provided")
  else
    sendToAIModel(inputText)
    generatedPaper = receiveAIResponse()

    saveToCloud(generatedPaper)
    display(generatedPaper)
  end if
End
```

MCQ Generation Algorithm
Input: Text / Topic
Output: MCQs with Answers
Start

```
   get textInput

   if textInput is empty then
      showMessage("Input text required")
   else
      sendToAIModel("Generate MCQs from text: " + textInput)
      mcqList = receiveAIResponse()

      saveToCloud(mcqList)
      displayMCQs(mcqList)
   end if
End
```

Summary Generation Algorithm

Input: Long Text

Output: Short Summary

```
Start
   get longText

   if longText is empty then
      showMessage("Enter text for summarization")
   else
      sendToAIModel("Summarize: " + longText)
      summary = receiveAIResponse()

      saveToCloud(summary)
      display(summary)
   end if
End
```

Citation / Reference Generation Algorithm

Input: Text, Selected Style (APA/MLA/Harvard)

Output: Text, Selected Style (APA/MLA/Harvard)

```
Start
   get textInput
   get citationStyle

   if textInput is empty then
      showMessage("No text found")
   else
      sendToAIModel("Generate " + citationStyle + " references for: " + textInput)
      citations = receiveAIResponse()

      saveToCloud(citations)
      display(citations)
   end if
End
```

File Download Algorithm (PDF or DOC)

Input: Generated Text

Output: File Saved to Device

```
Start
   get generatedText
   get selectedFormat

   if selectedFormat == PDF then
      createPDF(generatedText)
   else if selectedFormat == DOC then
      createDOC(generatedText)
   end if

   saveToDevice()
   showMessage("File downloaded successfully")
End
```

Chatbot Response Algorithm

Input: User Query

Output: AI Chatbot Response

```
Start
   get userQuery

   if userQuery is empty then
      showMessage("Please enter a query")
   else
      sendToAIModel("Answer this: " + userQuery)
      botResponse = receiveAIResponse()

      display(botResponse)
   end if
End
```

User Profile Update Algorithm

Input: Updated Name, Email, Image, etc.

Output: Updated Profile Successfully

```
Start
   get updatedProfileData

   if anyFieldEmpty() == true then
      showMessage("Fill all required fields")
   else
      saveToCloud(updatedProfileData)
      showMessage("Profile updated successfully")
   end if
End
```

Chapter 5

Testing and Evaluation

5. Introduction

To ensure the reliability, accuracy, and smooth performance of the AI Paper Generator application, a complete testing strategy was applied across all major modules, including File Uploading, AI Paper Generation, Summary Generation, MCQ Generation, Chatbot, and Exporting functionalities. Manual testing methods were used to validate user interactions, system responses, and AI-generated outputs. All test cases follow testing best practices: they are independent, clearly structured, avoid hardcoded values, and are repeatable for consistent verification. This systematic testing approach ensures that every feature of the AI Paper Generator app operates efficiently, securely, and provides a user-friendly experience.

5.1 Unit Testing (UT)

Tables of Unit Testing Test cases

1. UT #01 (User Login Functionality)

Test Case ID	UT_01
Requirement ID	RF_01
Title	User Login Functionality
Description	Testing the user login process in app
Objective	To ensure the user can successfully log in with valid credentials and receive appropriate errors for invalid data.
Driver/precondition	The user must already have an existing registered account.
Test steps	1. Open AI Paper Generator app 2. Navigate to Login screen 3. Enter email & password 4. Tap "Login"
Input	Valid email: user@example.com Valid password: Password123
Expected Results	The user successfully logs into the Home/Dashboard.
Actual Result	Successful login; Invalid credentials error; No internet.
Remarks	Pass

2. UT#02 (File Upload Functionality (PDF/DOC/TXT))

Test Case ID	UT_02
Requirement ID	RF_02
Title	File Upload and Text Extraction
Description	Testing file upload and text extraction processes.

Objective	To ensure the system uploads supported files and extracts readable text.
Driver/precondition	Users must be logged in. File must exist in device storage.
Test steps	1. Go to “Upload Paper” module 2. Select a PDF/DOC/TXT file 3. Tap "Upload"”.
Input	PDF file: sample.pdf
Expected Results	File uploads successfully and extracted text is displayed.
Actual Result	File uploaded; invalid file error shown; extraction successful.
Remarks	Pass

3. UT#03 (AI Paper Generation)

Test Case ID	UT_03
Requirement ID	RF_03
Title	Generate AI Research Paper
Description	Testing AI text generation module.
Objective	To ensure the system generates a complete and readable research paper based on input text or topic.
Driver/precondition	Input text must be provided by the user.
Test steps	1. Open “Generate Paper” screen 2. Enter topic or paste text 3. Tap "Generate"
Input	Topic: Artificial Intelligence in Education
Expected Results	AI generates structured paper sections (intro, body, conclusion).
Actual Result	Paper generated successfully; Model timeout observed.
Remarks	Pass

4. UT#04 (Summary Generation)

Test Case ID	UT_04
Requirement ID	RF_04
Title	Summary Generator
Description	Testing summarization feature.
Objective	To ensure the system generates a short and accurate summary of the provided text.
Driver/precondition	Users must upload or paste long text.
Test steps	1. Go to Summary Module

	2. Paste long text 3. Tap "Generate Summary"
Input	Paragraph (200–500 words)
Actual Result	Summary generated; No response on poor internet.
Remarks	Pass

5. UT#05 (MCQ Generation)

Test Case ID	UT_05
Requirement ID	RF_05
Title	MCQ Generator
Description	Testing the generation of multiple-choice questions.
Objective	To ensure the system correctly creates MCQs with correct answers from input text.
Driver/precondition	Users must provide some meaningful content.
Test steps	1. Open MCQ Generator 2. Paste/Upload content 3. Tap "Generate MCQs"
Input	Text content (150–300 words)
Expected Results	MCQs with 4 options and 1 correct answer.
Actual Result	MCQs generated; Some questions repeated on low data.
Remarks	90% Pass

5.2 Functional Testing (FT)

Tables of Functional Testing Test cases

1. FT#01 (User Registration)

Test Case ID	FT_01
Requirement ID	RF_01
Title	User Registration
Description	Test the functionality for a new user to register successfully.
Objective	Ensure users can create a new account with valid information.
Driver/precondition	AI Paper Generator app must be installed and launched.
Test steps	1. Open the app. 2. Tap “Sign Up”.

	3. Enter valid user details. 4. Tap “Register”.
Input	Email: testuser@example.com Password: Test@1234 Name: Test User
Expected Results	Account created, user moved to dashboard or login screen.
Actual Result	Account created successfully; redirected correctly.
Remarks	Pass

2. FT#02 (Upload File for Paper Generation)

Test Case ID	FT_02
Requirement ID	RF_02
Title	Upload File (PDF/DOC/TXT)
Description	Verify that the user can upload a supported file and the system extracts its content.
Objective	Ensure upload and text extraction work correctly.
Driver/precondition	Users must be logged in. The file must exist in storage.
Test steps	1. Navigate to “Upload Document”. 2. Select PDF/DOC/TXT file. 3. Tap upload. 4. Wait for extraction.
Input	sample.pdf
Expected Results	File uploaded and text is extracted successfully.
Actual Result	File uploaded and extraction successful.
Remarks	Pass

3. FT#03 (Generate Research Paper)

Test Case ID	FT_03
Requirement ID	RF_03
Title	Research Paper Generation
Description	Validate that the system generates a structured research paper from input.
Objective	Ensure the AI creates an Introduction, Body, and Conclusion successfully.
Driver/precondition	Users must provide topic or input text.

Test steps	1. Go to “Generate Paper”. 2. Enter topic or paste text. 3. Tap “Generate Paper”.
Input	Topic: Impact of AI in Healthcare
Expected Results	A complete research paper with structured sections is generated.
Actual Result	Research paper generated successfully.
Remarks	Pass

4. FT#04 (Generate MCQs)

Test Case ID	FT_04
Requirement ID	RF_04
Title	MCQ Generation
Description	Verify that the system generates meaningful multiple-choice questions.
Objective	Ensure MCQs are generated with 4 options and 1 correct answer.
Driver/precondition	Text must be uploaded or pasted.
Test steps	1. Open MCQ Generator. 2. Paste/upload content. 3. Tap “Generate MCQs”.
Input	Text about Machine Learning basics
Expected Results	The system generates 4–10 MCQs with correct answers.
Actual Result	MCQs generated successfully.
Remarks	Pass

5. FT#05 (Summary Generation)

Test Case ID	FT_05
Requirement ID	RF_05
Title	Summary Generator
Description	Test the summarization module for correctness.
Objective	Ensure concise and accurate summaries are produced.
Driver/precondition	Users must provide long text.
Test steps	1. Navigate to the Summary module. 2. Paste long text. 3. Tap “Generate Summary”.
Input	Paragraph (300 words)

Expected Results	A short and meaningful summary is displayed.
Actual Result	Summary generated successfully.
Remarks	90% Pass

5.3 Integration Testing (IT)

Tables of Integration Testing Test cases

1. IT#01 (Authenticate User and Access Document Upload Module)

Test Case ID	IT_01
Requirement ID	RF_01, RF_02
Title	Authenticate User and Access Document Upload Module
Description	Test the complete flow from user login to uploading a file, ensuring both modules integrate smoothly.
Objective	Verify that only authenticated users can access the Upload Module and that file upload works after login.
Driver/precondition	Users must have an existing account; app installed and launched.
Test steps	1. Open AI Paper Generator. 2. Enter valid login credentials. 3. Tap “Login”. 4. Navigate to “Upload Document”. 5. Select a PDF and upload it.
Input	Email: user@example.com Password: Test1234 File: research.pdf
Expected Results	The user is authenticated and successfully uploads the file; text extraction begins.
Actual Result	Same as expected.
Remarks	Pass

2. IT#02 (Upload Document and Generate Research Paper)

Test Case ID	IT_02
Requirement ID	RF_02, RF_03
Title	Upload Document and Generate Research Paper
Description	Test integration between the upload module and paper generation

	engine.
Objective	Ensure extracted content is passed correctly to the paper generator module.
Driver/precondition	Users must be logged in; file uploaded successfully.
Test steps	1. Upload a valid PDF. 2. Wait for extraction. 3. Navigate to “Generate Paper”. 4. Select extracted text. 5. Tap “Generate Paper”.
Input	File: ai_future.pdf
Expected Results	Paper generator receives extracted text; structured research paper is generated.
Actual Result	Works as expected.
Remarks	Pass

3. IT#03 (Generate Paper and Then Generate MCQs)

Test Case ID	IT_03
Requirement ID	RF_03, RF_05
Title	Generate Paper and Then Generate MCQs
Description	Check integration between Research Paper Generator and MCQ Generator.
Objective	Ensure MCQ generator uses generated paper content properly.
Driver/precondition	The user must have generated a research paper already.
Test steps	1. Generate research papers from input. 2. Tap “Generate MCQs”. 3. Confirm MCQ generation source as “Generated Paper”.
Input	Topic: Role of AI in Education
Expected Results	MCQs generated with relevant questions and correct options.
Actual Result	Same as expected.
Remarks	Pass

4. IT#04 (Generate Summary and Export Document)

Test Case ID	IT_04
Requirement ID	RF_04, RF_06
Title	Generate Summary and Export Document
Description	Validate integration between the Summary Generator and Export

	Module.
Objective	Ensure generated summary can be exported as PDF/DOC format.
Driver/precondition	Summary must already be generated.
Test steps	1. Navigate to Summary Generator. 2. Generate summary from large text. 3. Tap “Export”. 4. Choose PDF/DOC. 5. Save file.
Input	500-word paragraph
Expected Results	Summary is exported successfully in selected file format.
Actual Result	Exported without errors.
Remarks	Pass

5. IT#05 (Multi-Module Workflow)

Test Case ID	IT_05
Requirement ID	RF_02, RF_03, RF_04, RF_06
Title	Upload → Generate Paper → Generate Summary → Export
Description	Test the end-to-end workflow across four integrated modules.
Objective	Ensure smooth transition and shared data flow among modules.
Driver/precondition	User logged in.
Test steps	1. Upload PDF. 2. Extract content. 3. Generate paper. 4. Generate summary. 5. Export summary as PDF.
Input	File: ML_research.pdf
Expected Results	The entire workflow executes smoothly without system errors.
Actual Result	Successful.
Remarks	Pass

5.5 Summary

This chapter presents an overview of the testing phase for the AI Paper Generator, focusing on ensuring reliability, accuracy, and smooth operation across all major modules. The testing process included Unit Testing, Functional Testing, and Integration Testing, each targeting different layers of the application.

Unit testing verified that individual components—such as text generation, topic selection, user authentication, PDF export, and reference generation—worked correctly in isolation. Functional testing validated whether end-to-end user actions (e.g., generating a research paper, selecting templates, downloading files) met the system requirements. Integration testing ensured proper interaction between interconnected modules, such as linking topic input with AI generation, referencing, formatting, and final document export.

By applying structured, repeatable, and clearly defined test procedures, the overall quality, stability, and user experience of the AI Paper Generator were significantly improved. This comprehensive testing approach ensures accuracy, usability, and trust in the final system, enabling users to generate academic papers efficiently and reliably.

Chapter 6

System Conversion

6. Introduction

System conversion refers to the process of transitioning from an old system to a new one. It includes transferring operations, workflows, and data to ensure the new system functions smoothly. Different conversion methods—such as direct, parallel, pilot, and phased—are commonly used depending on system complexity, modules, and risk level.

Since AI Paper Generator involves multiple interconnected components (AI generation, formatting, export, referencing, templates, etc.), a structured approach was required to minimize errors and ensure stable deployment.

6.1 Conversion Method

For the **AI Paper Generator**, the **Phased Conversion** method was selected.
This method is ideal because:

- The system contains many independent modules.
- Each feature can be deployed, tested, and improved step by step.
- Risks are minimized since each phase is validated before moving to the next.
- Users can gradually adapt to new functionalities.

6.1.1 Selected Method: Phased Conversion

Phased conversion was used to implement and release modules in a controlled, sequential manner. The system is divided into phases, with each phase focusing on specific sections of the application. After completing a phase, testing and validation take place before moving forward. This improves stability, makes bug fixes easier, and ensures smooth system growth.

Here is the table of all phase of development in AI Paper Generator: -

Phase	Module Names	Purpose
Phase 1	SplashScreen, Login/Signup, User Profile Setup	Establish basic structure, allow user entry, manage authentication, and collect user preferences.
Phase 2	Topic Input Module, Paper Type Selection (Essay, Assignment, Research Paper)	Allow users to select the type of paper and provide the topics or instructions.
Phase 3	AI Content Generation Engine	Generate introduction, body, conclusion, and smart content blocks using NLP models.

Phase 4	Formatting & Structuring Module	Convert generated text into academic formats (APA/MLA/IEEE), headings, spacing, alignment, and layout.
Phase 5	Reference Generator, Citations Manager	Automatically generate and insert citations and reference lists in the selected style.
Phase 6	Export Module (PDF, DOCX, Text)	Allow users to download the final paper in multiple formats with proper formatting.
Phase 7	Templates Library & Customization	Provide pre-built academic templates and allow users to customize layouts and structure.
Phase 8	AI Revision Assistant & Plagiarism Checker	Provide improvements, rewrites, grammar fixes, and detect similar content.
Phase 9	Cloud Storage Sync	Allow users to save, restore, and manage generated papers online for future access.

Chapter 7 Conclusion

7. Introduction

This chapter concludes the AI Paper Generator project by evaluating the achievements, analyzing completed and pending objectives, mapping requirements with design and testing through a traceability matrix, and presenting the final conclusion along with future enhancements. The chapter also provides clarity on how the system evolved from requirements to implementation and how well it met its intended goals.

7.1 Evaluation

The following table presents the list of objectives defined at the start of the project along with their current completion status:

Objectives	Status
User Login and Basic Profile Setup	Completed
Topic Input and Paper Type Selection	Completed
AI Generation for Academic Papers	Completed
Automatic Formatting (APA/MLA/IEEE)	Completed
Reference & Citation Auto-Generator	Completed
Export in PDF, DOCX, and Text Formats	Completed
Templates Library for Academic Papers	Completed
AI Revision Assistant (Rewrite/Improve Content)	Completed
Plagiarism Checking System	Completed
Cloud Storage for Saving Papers	Completed

User Dashboard to Manage Documents	Completed
Settings & Profile Management	Completed

7.2 Traceability Matrix

Table: Traceability Matrix

Requirement ID	Requirement Description	Design Specification	Code (Frontend / Backend)	Test IDs
RF_01	User Registration / Login	Authentication Module	Frontend: `Login.jsx`, `Signup.jsx` Backend: `auth_api.py` (FastAPI)	UT_01, FT_01, IT_01
RF_02	AI Paper Generation	AI Generation Engine	Frontend: `PaperGenerator.jsx`, `UploadDocument.jsx` Backend: `ai_generator.py` (FastAPI worker / service)	UT_02, FT_02, IT_02
RF_03	Formatting (APA, MLA, IEEE)	Formatting Module	Frontend: `FormattingOptions.jsx` Backend: `formatter.py` (formatting utilities / services)	UT_03, FT_03
RF_04	Citation & References	Citation Manager	Frontend: `ReferencesPanel.jsx` Backend: `reference_generator.py` (citation engine)	UT_04, FT_04

RF_05	Export Features (PDF, DOCX, Text)	Export Module	Frontend: `ExportDialog.jsx` Backend: `export_handler.py` (PDF/DOCX generation)	UT_05, FT_05
RF_06	Templates Library	Templates Component	Frontend: `TemplateLibrary.jsx`, `TemplateEditor.jsx` Backend: `template_manager.py` (template storage & rendering)	UT_06
RF_07	AI Revision Assistant	Content Improvement Module	Frontend: `RevisionAssistant.jsx` Backend: `ai_rewriter.py` (rewriter service)	UT_07, IT_03
RF_08	Cloud Storage Integration	Cloud Module	Frontend: `CloudSyncUI.jsx` Backend: `cloud_sync.py` (cloud save/load APIs)	IT_04
RF_09	Document Dashboard	Dashboard UI Module	Frontend: `Dashboard.jsx`, `DocumentList.jsx` Backend: `dashboard_api.py` (document management endpoints)	FT_06

7.3 Conclusion

In conclusion, the **AI Paper Generator** successfully meets all major project objectives. The system provides a complete, automated, and user-friendly platform for generating academic papers. It reduces the workload of students, researchers, and professionals by automating:

- Content writing
- Formatting
- Citation generation

- Structuring
- Exporting
- Improvement & proofreading

All essential functionalities, such as topic input, paper generation, citation management, formatting, templates, cloud storage, and export options, were implemented effectively and tested for quality.

Objective and Functionality Table

Objective	Status	Functional Module(s) Implemented
Generate academic papers using AI	Completed	AI Content Generator
Format papers in APA/MLA/IEEE styles	Completed	Formatting Module
Provide automatic references & citations	Completed	Citation Manager
Export in multiple formats	Completed	PDF/DOCX/Text Export Module
Allow topic input & paper type selection	Completed	Topic Input Module
Improve content with AI rewriting	Completed	AI Revision Assistant
Store papers securely online	Completed	Cloud Storage Module
Provide templates for structured papers	Completed	Templates Library

Manage all generated papers via dashboard	Completed	Document Dashboard
Enable user login & profile setup	Completed	Authentication Module

Future Work

- Integration with **Google Scholar API** for automatic reference fetching.
- Adding **voice-based topic input** for faster user interaction.
- Enhanced plagiarism checker using advanced algorithms.
- Multi-language paper generation (e.g., Urdu, German, Arabic).
- Adding **collaboration mode**, allowing multiple users to edit the same document.
- Offline mode for generating content without the internet.
- AI-powered **outline generator** to help users plan structure before writing.

Glossary

AI Paper Generator
The complete system that ingests documents, analyzes content, generates questions, creates assessments, and exports structured outputs using AI models and vector search.

Ingestion Pipeline
The module responsible for accepting PDFs or text files, extracting readable content, cleaning text, and producing normalized chunks ready for indexing.

OCR (Optical Character Recognition)
A technique used to extract text from scanned or image-based PDFs. Applied automatically when PDF text extraction fails.

Chunk / Semantic Chunking
Small, coherent text segments (paragraphs or token-sized windows) created during preprocessing to enable retrieval, embedding, and question generation.

Embedding
A numerical vector representation of text used for similarity search and retrieval. Generated using Gemini or Opus embedding models.

Qdrant
A high-performance vector database used to store embeddings and perform semantic search (top-K retrieval). Stores metadata like doc_id, page number, and chunk text.

Retrieval-Augmented Generation (RAG)
A method where the system fetches the most relevant text chunks from Qdrant and feeds them into the LLM to produce grounded, hallucination-free questions.

Topic Analyzer
The module that clusters embedded chunks to identify major themes, keywords, and hierarchical topic structures across uploaded documents.

MCQ Generator (Question Generator)
The AI engine that produces multiple-choice questions based on retrieved context. Generates the question stem, correct answer, and plausible distractors.

Distractor
An incorrect but believable answer choice used in MCQs. Evaluated for plausibility, semantic distance, and grammatical consistency.

Distractor Validator
The subsystem that ensures distractors are not too similar or too different from the correct answer, preventing trivial or misleading options.

Bloom's Taxonomy
A classification used to tag cognitive levels of questions (remember, understand, apply, analyze, evaluate, create).

Difficulty Level
Categorization of questions into Easy, Medium, or Hard based on linguistic complexity, reasoning steps, and content depth.

Assessment Builder
Module that assembles validated questions into structured outputs such as quizzes, assignments, and exams with customizable parameters.

Grounding Check
A validation process confirming that the correct answer is supported by retrieved source text and not hallucinated.

Quality Score
A numerical rating assigned to each generated question based on grounding, clarity, distractor quality, and uniqueness.

JSON Output Schema
A structured format containing generated questions, assessments, and metadata. Required for final submission and evaluation.

Cloud Storage Integration
Optional component allowing users to store or retrieve documents and generated outputs using cloud services.

Frontend / UI
User-facing interface (React) that supports uploading PDFs, viewing topics, reviewing questions, and exporting assessments.

Backend / API Services
Python services (FastAPI) responsible for orchestration, Qdrant queries, LLM calls, validation checks, and exporting final JSON.

Evaluation Pipeline
Testing process that measures grounding rate, quality score, difficulty accuracy, and human acceptance of generated questions.

Human-in-the-Loop Review
Optional review step where users manually approve or edit AI-generated questions before finalizing assessments.

Export Module
Component that outputs files such as JSON, CSV, PDF, or DOCX for quizzes, assignments, and exams.

Orchestration Layer

A system that handles batching, scheduling, retries, and cost control for large-scale embedding and question-generation tasks.