"DocuCore: Text & Code Summariser in Cloud"

A

Project Report

submitted

in partial fulfillment

for the award of the Degree of

Bachelor of Technology

in Department of Information Technology

Project Mentor:
Mr. Praveen Kumar Yadav
Assistant Professor

Submitted By :
Abhishek Gupta(21ESKIT003)
Hiteshi Agrawal(21ESKIT056)
Garvita Sakhrani(21ESKIT049)

Department of Information Technology
Swami Keshvanand Institute of Technology, M & G, Jaipur
Rajasthan Technical University, Kota
Session 2024-2025

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

Department of Information Technology

CERTIFICATE

This is to certify that Ms. Garvita Sakhrani, a student of B.Tech(Information Technology) VIII semester has submitted her Project Report entitled "DocuCore: Text & Code Summariser in Cloud" under my guidance.

Mentor

Mr. Praveen Kumar Yadav

Assistant Professor

Signature.....

Coordinator

Dr. Priyanka Yadav

Assistant Professor

Signature.....

Swami Keshvanand Institute of Technology, Management & Gramothan, Jaipur

Department of Information Technology

CERTIFICATE

This is to certify that Mr. Abhishek Gupta, a student of B.Tech(Information Technology) VIII semester has submitted his Project Report entitled "DocuCore: Text & Code Summariser in Cloud" under my guidance.

Mentor	**Coordinator**
Mr. Praveen Kumar Yadav	Dr. Priyanka Yadav
Assistant Professor	Assistant Professor
Signature.....	Signature.....

Swami Keshvanand Institute of Technology,

Management & Gramothan, Jaipur

Department of Information Technology

CERTIFICATE

This is to certify that Ms. Hiteshi Agrawal, a student of B.Tech(Information Technology) VIII semester has submitted his Project Report entitled "DocuCore: Text & Code Summariser in Cloud" under my guidance.

Mentor	**Coordinator**
Mr. Praveen Kumar Yadav	Dr. Priyanka Yadav
Assistant Professor	Assistant Professor
Signature.....	Signature.....

DECLARATION

We hereby declare that the report of the project entitled ”DocuCore: Text & Code Summariser in Cloud” is a record of an original work done by us at Swami Keshvanand Institute of Technology, Management and Gramothan, Jaipur under the mentorship of Mr. Praveen Kumar Yadav(Dept. of Information Technology) and coordination of Dr. Priyanka Yadav(Dept.of Information Technology). This project report has been submitted as the proof of original work for the partial fulfillment of the requirement for the award of the degree of Bachelor of Technology (B.Tech) in the Department of Information Technology. It has not been submitted anywhere else, under any other program to the best of our knowledge and belief.

Team Members **Signature**

(Garvita Sakhrani, 21ESKIT049)

(Hiteshi Agrawal, 21ESKIT056)

(Abhishek Gupta, 21ESKIT003)

Acknowledgement

A project of such a vast coverage cannot be realized without help from numerous sources and people in the organization.We take this opportunity to express our gratitude to all those who have been helping us in making this project successful.

We are highly indebted to our faculty mentor Mr. Praveen Kumar Yadav .He has been a guide, motivator, source of inspiration for us to carry out the necessary proceedings for the project to be completed successfully. We also thank our project coordinator Dr. Priyanka Yadav for her co-operation, encouragement, valuable suggestions and critical remarks that galvanized our efforts in the right direction.

We would also like to convey our sincere thanks to Dr. Anil Chaudhary, HOD, Department of Information Technology, for facilitating, motivating and supporting us during each phase of development of the project.Also, we pay our sincere gratitude to all the Faculty Members of Swami Keshvanand Institute of Technology, Management and Gramothan, Jaipur and all our Colleagues for their co-operation and support.

Last but not least we would like to thank all those who have directly or indirectly helped and cooperated in accomplishing this project.

Team Members:
(Garvita Sakhrani, 21ESKIT049)
(Hiteshi Agrawal, 21ESKIT056)
(Abhishek Gupta, 21ESKIT003)

Abstract

This project introduces a cloud-based system designed for document summarization and code analysis, utilizing advanced machine learning technologies, particularly Large Language Models (LLMs). The goal is to provide an efficient, scalable solution for processing large volumes of text and code.

The Text Summarization Module automatically extracts key insights from lengthy documents, helping users quickly grasp important information. This feature is valuable for summarizing reports, articles, and technical papers in a concise and accurate manner.

The Code-Based Analysis Module automates the process of code review, offering suggestions to improve code quality and performance. This module supports multiple programming languages and is aimed at streamlining software development workflows.

By deploying the system on the cloud, it ensures scalability, real-time processing, and efficient collaboration. The integration of cloud technology allows the system to dynamically scale resources based on user demands, making it a flexible and powerful tool for modern data and software analysis needs.

Table of Content

1 Introduction 1

1.1 Problem Statement and Objective 1

1.2 Literature Survey /Market Survey/Investigation and Analysis 1

1.3 Introduction to Project 2

1.4 Proposed Logic / Algorithm / Business Plan / Solution / Device 3

1.4.1 Logic 3

1.4.2 Algorithm 3

1.4.3 Solution 3

1.5 Scope of the Project 3

2 Software Requirement Specification 5

2.1 Overall Description 5

2.1.1 Product Perspective 5

2.1.1.1 System Interfaces 5

2.1.1.2 User Interfaces 6

2.1.1.3 Hardware Interfaces 6

2.1.1.4 Software Interfaces 7

2.1.1.5 Communications Interfaces 7

2.1.1.6 Memory Constraints 7

2.1.1.7 Operations 8

2.1.1.8 Project Functions 8

2.1.1.9 User Characteristics 8

2.1.1.10 Constraints 9

2.1.1.11 Assumption and Dependencies 9

3 System Design Specification 11

3.1 System Architecture 11

3.1.0.1 Presentation Layer 11
3.1.0.2 API Layer 11
3.1.0.3 Processing Layer 11
3.1.0.4 Data Layer 12
3.1.0.5 DevOps & CI/CD 12
3.1.0.6 Monitoring & Logging 12
3.2 Module Decomposition Description 12
3.3 High Level Design Diagrams 15
3.3.1 Use Case Diagram 15
3.3.2 Activity Diagram 16
3.3.3 Data-Flow Diagram 17
3.3.4 ER Diagram 18
3.3.5 Class Diagram 20
3.3.6 Communication Diagram 20
3.3.7 Sequence Diagram 20
3.3.8 Component Diagram 21
3.3.9 Deployment Diagram 22
3.3.10 Business Process Model 23

4 Methodology and Team 26
4.1 Introduction to Waterfall Framework 26
4.2 Team Members, Roles & Responsibilities 28

5 Centering System Testing 29
5.1 Functionality Testing 29
5.2 Performance Testing 30
5.3 Usability Testing 30

6 Test Execution Summary 32

7 Project Screenshots 34

8 Project Summary and Conclusions **39**

8.1 Project Summary 39

8.2 Conclusion 40

9 Future Scope **41**

References **42**

List of Figures

3.1 Architecture Diagram 15
3.2 Use Case Diagram 15
3.3 Activity Diagram 16
3.4 Data Flow Diagram (DFD) Level 0 17
3.5 Data Flow Diagram (DFD) Level 1 18
3.6 Data Flow Diagram (DFD) Level 2 18
3.7 ER Diagram 19
3.8 Class Diagram 20
3.9 Communication Diagram 20
3.10 Sequence Diagram 21
3.11 Component Diagram 22
3.12 Deployment Diagram 23
3.13 Business Process Model 25

4.1 WaterFall model 26

7.1 Home Interface Displaying Access to Summarisation and Code Analysis Modules 34
7.2 Deployment Interface 35
7.3 Source Code Analysis Dashboard 35
7.4 Agent Pools 36
7.5 Chatbot App Dashboard 36
7.6 Databricks V.M. Configuration 37

List of Tables

6.1 Test Execution Summary Table 33

Chapter 1
Introduction

1.1 Problem Statement and Objective

In today's fast-paced development environment, developers often face significant challenges in comprehending large codebases and navigating through lengthy technical documents. This complexity leads to reduced productivity, slower onboarding of new team members, and an increased risk of introducing errors into systems.

The objective of this project is to automate and streamline the analysis of text documents and source code by leveraging cloud technologies. A key focus is on designing robust, scalable automation pipelines that can efficiently process, parse, and extract meaningful insights from massive volumes of data. These pipelines will eliminate the need for manual review, ensuring faster analysis, minimizing human errors, and enabling developers to focus more on innovation rather than routine comprehension tasks.

By integrating intelligent automation into the workflow through cloud-based pipelines, the system aims to revolutionize how textual and code-based information is consumed, interpreted, and acted upon.

1.2 Literature Survey /Market Survey/Investigation and Analysis

Several tools currently assist developers in improving code quality and managing large documents. SonarQube and ESLint are widely used for static code analysis, helping to identify bugs, maintain coding standards, and improve code readability. Similarly, Large Language Models (LLMs) like GPT have shown strong capabilities

in tasks such as text summarization, documentation analysis, and natural language understanding.

However, most of these tools operate independently and are focused on specific tasks. SonarQube and ESLint primarily address source code quality, while LLMs focus on text and document analysis. There is no common platform that integrates both capabilities into a single automated system.

Additionally, the integration of these tools into a real-time cloud-based platform with CI/CD (Continuous Integration and Continuous Deployment) pipelines remains largely unexplored. Existing solutions do not provide a seamless workflow where code analysis, document understanding, and automated deployment are combined together. As a result, developers still have to manage multiple tools separately, leading to fragmented workflows and additional manual effort.

This highlights a clear need for a unified platform that can automate both code and text analysis within a cloud-based CI/CD environment, improving efficiency, accuracy, and overall productivity.

1.3 Introduction to Project

DocuCore is a cloud-based platform designed to automate both code analysis and document summarization. It leverages services like Azure, Databricks, and CI/CD pipelines to create a seamless, real-time workflow. By integrating these technologies, DocuCore ensures faster, accurate processing of large codebases and technical documents. Unlike existing tools, it provides a unified solution where automation reduces manual effort and enhances developer productivity. The system addresses the lack of integrated platforms by combining static analysis, document understanding, and automated deployment in a single cloud environment.

1.4 Proposed Logic / Algorithm / Business Plan / Solution / Device

1.4.1 Logic

The system follows a simple yet effective workflow:
Upload → Summarize/Analyze → Store → Display.
Users upload codebases and documents, which are then automatically processed for analysis and summarization, stored securely, and made accessible through a user-friendly interface.

1.4.2 Algorithm

- **Summarisation:** Utilizes Large Language Models (LLMs) for automatic text summarization and document understanding.
- **Code Analysis:** Employs static analysis tools to identify code quality issues, vulnerabilities, and improvement opportunities.

1.4.3 Solution

The proposed system is a cloud-hosted, real-time, and scalable platform. It integrates services like Azure, Databricks, and CI/CD pipelines to ensure continuous automation, reliability, and efficient processing of large-scale documents and source code.

1.5 Scope of the Project

The scope of this project is to develop a cloud-based automated system that integrates code analysis and document summarization on a single platform. By leveraging services such as Azure, Databricks, and CI/CD pipelines, the system aims to provide real-time processing, scalable deployment, and continuous improvement of large codebases and technical documents.
The platform will support:

- Automated uploading, analysis, storage, and display of both code and textual data.
- Static analysis tools for evaluating code quality, security, and maintainability.
- Large Language Models (LLMs) for intelligent summarization and understanding of technical documents.
- Real-time cloud integration ensuring high availability, scalability, and seamless user experience.
- CI/CD automation for continuous integration, testing, and deployment without manual intervention.

The project is intended to streamline developer workflows, reduce manual efforts, enhance productivity, and ensure consistent quality across software development and documentation processes. It also opens avenues for future enhancements like multi--language support, AI-driven recommendations, and advanced security integrations.

Chapter 2
Software Requirement Specification

2.1 Overall Description

The system, DocuCore: Text & Code Summariser in Cloud, is a cloud-based application designed to automate document summarization and code analysis using state-of-the-art tools and platforms. It leverages Azure Cloud, Databricks, Git, CI/CD pipelines, and modern code management platforms to provide scalable, efficient, and real-time services.

2.1.1 Product Perspective

The system is a new, independent cloud application that provides:

- Summarization of uploaded documents (e.g., research papers, articles).
- Static analysis of code files to detect bugs, vulnerabilities, and improvements.
- Deployment and hosting on Azure for scalability, security, and high availability.

It interacts with users through a user-friendly web interface and integrates with Azure backend services through RESTful APIs and secure communication protocols.

2.1.1.1 System Interfaces

- **Azure Services:** Backend is hosted on Azure services such as Azure Functions, Azure Storage, and Databricks.
- **Azure DevOps Pipelines:** For CI/CD operations, managing deployments and updates.

- **Databricks Workspace:** For handling LLM summarization workflows and code analysis tasks.
- **Git Repositories:** Source code and scripts are version-controlled using Git.
- **Monitoring Tools:** Azure Monitor and Log Analytics for system health monitoring.

2.1.1.2 User Interfaces

- **Web Application (Frontend):**
  A simple and responsive web interface where users can upload documents (.pdf, .docx, .txt) and code files (.py, .java, .js).
  The UI is built for easy navigation and fast interactions, ensuring that users can quickly upload files without any complications.
- **Result Display Dashboard:**
  Once processing is complete, users can immediately view the generated document summaries or code analysis reports in a clean and organized layout.
- **Upload Status Notifications:**
  The interface provides real-time feedback about file upload progress, processing status, and completion notifications.

2.1.1.3 Hardware Interfaces

- **User Devices:**
  Standard laptops, desktops, or mobile devices with internet connectivity and a modern browser (Chrome, Edge, Firefox).
- **Cloud Infrastructure:**
  Azure-hosted Virtual Machines (VMs), Databricks clusters, and storage accounts.

2.1.1.4 Software Interfaces

- **Azure Functions and Databricks Notebooks:** Serverless code execution and LLM processing.
- **GitHub/Azure Repos:** For source control and version management.
- **VSCode:** Used as the primary IDE for development and debugging.
- **Databricks App:** For code quality analysis & text summarisation during CI/CD pipelines.

2.1.1.5 Communications Interfaces

- **WRESTful APIs:**
  Used for communication between the web frontend and backend services, ensuring seamless document upload, processing, and result retrieval.
- **HTTPS Protocol:**
  All data transmissions between the client (user browser) and server (Azure-hosted services) are secured using HTTPS, ensuring encryption and protection against attacks.
- **Git over SSH:**
  Secure transfer of source code between local development machines, GitHub, and Azure Repos, using SSH keys for authentication. **Microsoft Self-Hosted Agent Communication:**
  The CI/CD pipelines use a Microsoft Self-Hosted Agent installed on a dedicated VM or machine. The agent securely communicates with Azure DevOps Services over HTTPS, allowing controlled build, test, and deployment tasks without exposing private infrastructure to the public internet.

2.1.1.6 Memory Constraints

- Minimum memory requirement: 8GB RAM for smooth processing.

- Databricks clusters dynamically manage memory based on workload.
- Uploaded files are limited to 50MB to optimize processing times.

2.1.1.7 Operations

- **Uploads:** Upload Documents/Code Files via Web UI
- **Backend Processing:** Summarization or Code Analysis triggered automatically.
- **Result Generation:** Summaries or reports generated and displayed to the user.
- **Continuous Deployment:** New updates rolled out via Azure DevOps pipelines without downtime.
- **Monitoring & Logging:** Real-time tracking using Azure Monitor and Application Insights.

2.1.1.8 Project Functions

- Document Summarization using LLM models on Databricks.
- Code Static Analysis using integrated tools.
- File Storage on Azure Blob Storage.
- Real-time Notifications for success/failure of uploads and processing.
- CI/CD based code deployments ensuring fast updates and rollbacks if needed.
- Version Control with Git integrated with Azure Repos.

2.1.1.9 User Characteristics

- **Primary Users:** Researchers, developers, students, and IT professionals.
- **Skill Level:** Basic understanding of file formats and web usage.
- **Expected Behavior:** Users should be able to upload documents/code, interpret reports, and download results.

- **Security Awareness:** Users are expected to maintain data privacy for their uploads.

2.1.1.10 Constraints

- System highly depends on the availability of Azure services.
- Upload size limit is 50MB per document/code file.
- System is optimized for modern web browsers (Chrome, Edge, Firefox) only.
- Resource Usage and Cost must be managed carefully due to cloud billing constraints.

2.1.1.11 Assumption and Dependencies

Assumptions

- **Internet Connectivity:** It is assumed that users will have a stable and fast internet connection to interact with the system, as the document uploads, code analysis, and cloud-based operations rely on internet connectivity.
- **User Technical Knowledge:** It is assumed that the users (researchers, developers, etc.) have basic knowledge of uploading documents or code files and interpreting the results provided by the system (e.g., summaries or code analysis reports).
- **Availability of AWS Services:** The system assumes that AWS services, such as Lambda, API Gateway, and Elastic Beanstalk, are available and functioning properly. Any service downtime could impact the system's ability to process requests in real-time.
- **Document and Code Formats:** It is assumed that the uploaded documents will be in standard formats like .txt, .pdf, or .docx, and the code files will follow commonly used programming languages.

Dependencies

- **Summarisation Modules:** The system depends on various modules for generating document summaries. If the API service experiences any downtime or changes in access policies, it may disrupt the summarisation functionality.
- **Code Analysis Tools:** The code analysis module relies on code analysis tools to evaluate uploaded code files for bugs, complexity, and vulnerabilities. The performance of the system is dependent on the availability and accuracy of these tools.
- **Cloud Infrastructure (AWS):** The entire system is hosted on AWS, meaning it depends on AWS services like S3 for storage, EC2/Lambda for compute resources. Any issues with these services could lead to performance degradation or downtime.
- **Browser Compatibility:** The system's web-based interface depends on modern web browsers for proper functioning. It assumes compatibility with popular browsers like Chrome, Firefox, and Edge.

Chapter 3
System Design Specification

3.1 System Architecture

The DocuCore platform follows a layered, micro-services-oriented architecture deployed entirely on Microsoft Azure. This design maximizes scalability, maintainability, and fault tolerance:

3.1.0.1 Presentation Layer

- **Web UI:** A single-page application built with React/TypeScript, served via Azure Static Web Apps.
- **Responsibilities:** File upload components, progress indicators, and result dashboards.

3.1.0.2 API Layer

- **Azure API Management:** Exposes versioned RESTful endpoints (`/upload`, `/summarize`, `/analyze`, `/status`).
- **Security:** HTTPS only, with CORS policies restricted to the official UI domain.

3.1.0.3 Processing Layer

- **Azure Functions:**
  - **Upload Handler:** Validates and stores incoming files, then emits metadata events.
  - **Summarization Worker:** Processes events to run LLM summarization jobs on Azure Databricks.

  - **Analysis Worker:** Executes static code analysis (e.g., SonarQube scanners) on uploaded code.

- **Azure Databricks:** Hosts notebooks and orchestrates GPT-based summarization workflows and embedding management.

3.1.0.4 Data Layer

- **Azure Blob Storage:** Persists raw documents, code files, generated summaries, and analysis reports.
- **ChromaDB (on Azure):** Manages embeddings for fast similarity search.
- **Azure SQL Database:** Stores job metadata, statuses, and audit logs.

3.1.0.5 DevOps & CI/CD

- **Azure DevOps Pipelines (Self-Hosted Agent):**
  - **Build Stage:** Lints and builds UI code; packages Function apps.
  - **Deploy Stage:** Deploys UI to Static Web Apps, publishes Functions, and updates Databricks jobs.

3.1.0.6 Monitoring & Logging

- **Azure Monitor & Application Insights:** Tracks execution times, error rates, and latency.
- **Log Analytics:** Provides centralized, queryable logs for forensic analysis.

3.2 Module Decomposition Description

The system is logically decomposed into the following modules. Each module encapsulates a clear set of responsibilities and exposes well-defined interfaces:

1. **UI Module**
   - Renders file-upload form and progress indicators.

- Displays document summaries and code analysis reports.
- Invokes REST API endpoints (`/upload`, `/status`, `/results`).

2. **API Gateway Module**
    - Routes incoming HTTP requests to appropriate backend services.
    - Enforces HTTPS, CORS, and rate-limiting policies.
    - Provides OpenAPI/Swagger documentation for all endpoints.
3. **Upload Handler Module**
    - Validates file type and size constraints.
    - Persists raw files into Azure Blob Storage.
    - Emits events to Azure Event Hubs for downstream processing.
4. **Summarization Module**
    - Consumes upload events from Azure Event Hubs.
    - Orchestrates LLM summarization jobs on Azure Databricks.
    - Stores generated summaries back into Blob Storage.
5. **Analysis Module**
    - Listens for code-upload events from Event Hubs.
    - Executes static code analysis (e.g., SonarQube) on the uploaded code.
    - Generates and persists detailed analysis reports.
6. **Storage Module**
    - Provides an abstraction layer over Azure Blob Storage and Azure SQL Database.
    - Implements CRUD operations for files, summaries, reports, and metadata.
7. **DevOps Module**
    - Defines Azure DevOps YAML pipelines for build, test, and deployment stages.

- Manages the Microsoft Self-Hosted Agent configuration and secure access keys.

8. **Monitoring Module**
    - Integrates Application Insights for telemetry capture.
    - Sets up alerts and dashboards in Azure Monitor for SLA and health monitoring.

Interaction Flow:

1. The user selects a file in the UI Module and clicks "Upload."
2. The UI Module sends the file to the API Gateway via HTTPS.
3. The API Gateway forwards the request to the Upload Handler, which stores the file and emits an event to Azure Event Hubs.
4. The Summarization and Analysis Modules consume the event, process the file, and store results via the Storage Module.
5. The UI Module polls the `/status` endpoint (or subscribes to a WebSocket) and retrieves completed summaries/reports for display.

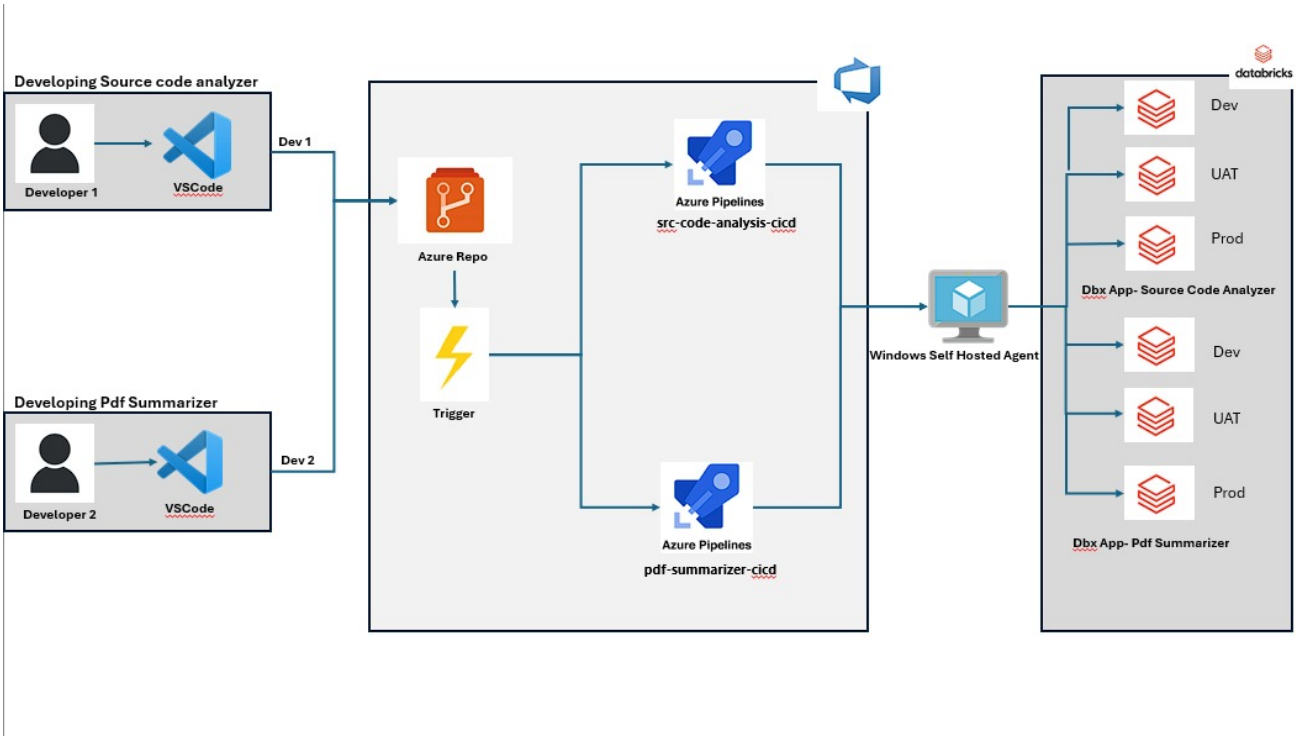

Figure 3.1: Architecture Diagram

3.3 High Level Design Diagrams

3.3.1 Use Case Diagram

The Use Case Diagram represents the interactions between the actors (users and system components) and the use cases (specific functionalities of the system).

Actors:

- **End User:** The user who uploads documents and code files for summarization or analysis.
- **Admin:** Manages the system, users, and database.

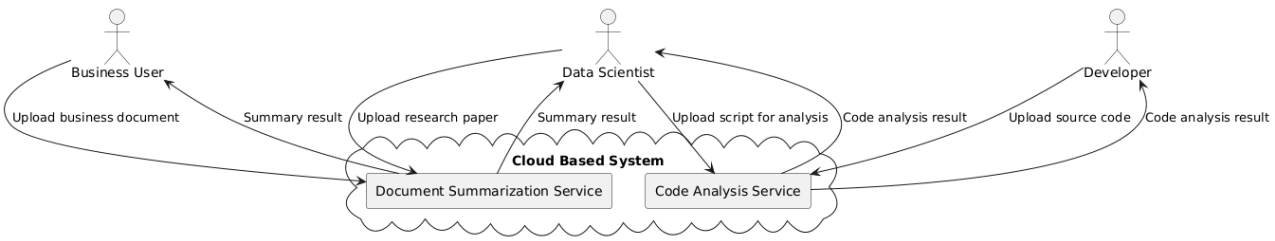

Figure 3.2: Use Case Diagram

3.3.2 Activity Diagram

The activity diagram outlines the steps involved in the summarization and analysis processes:

- **Upload Action:** The user initiates the process by uploading a document.
- **Data Processing:** The system processes the uploaded file using the appropriate module (LLM or code analysis).
- **Result Generation:** Summaries or code analysis reports are generated.
- **Storage:** The results are stored in the cloud (e.g., Azure Blob Storage).
- **Result Delivery:** The results are returned to the user for viewing.

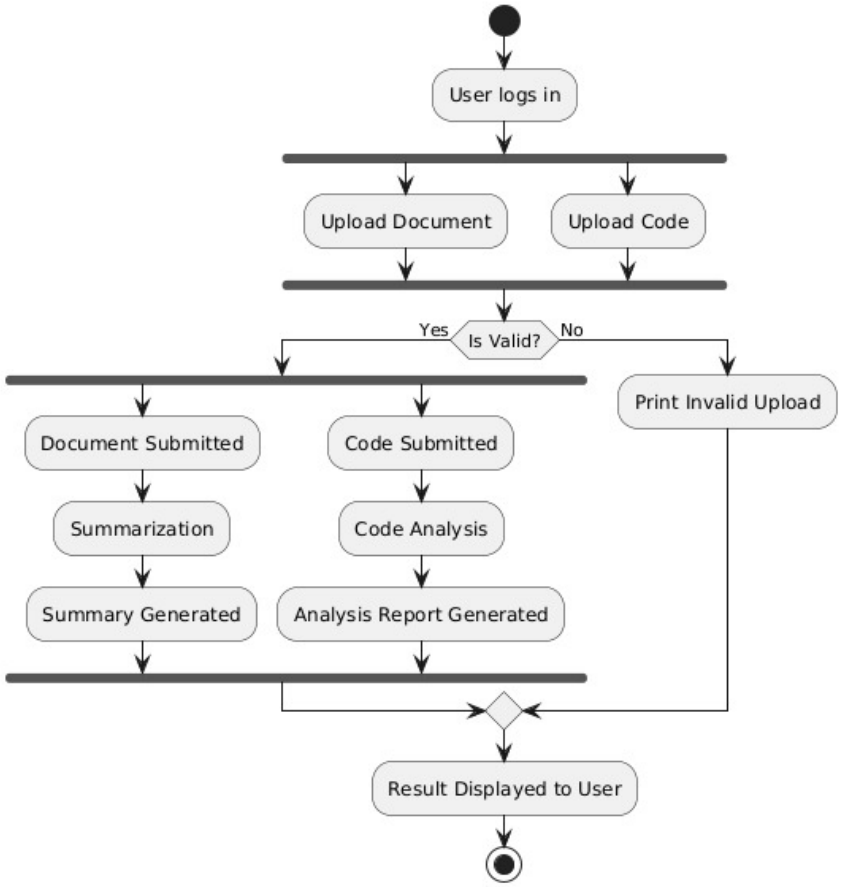

Figure 3.3: Activity Diagram

3.3.3 Data-Flow Diagram

The data flow diagram outlines how data moves between various components of the system:

- **User Inputs:** Users upload documents through the web interface.
- **Data Processing:** The system routes the inputs to LLMs for document summarization or code analysis tools for processing code.
- **Storage:** Processed summaries and analysis reports are stored in AWS S3.
- **Output:** The summarized documents and analysis reports are sent back to the user.

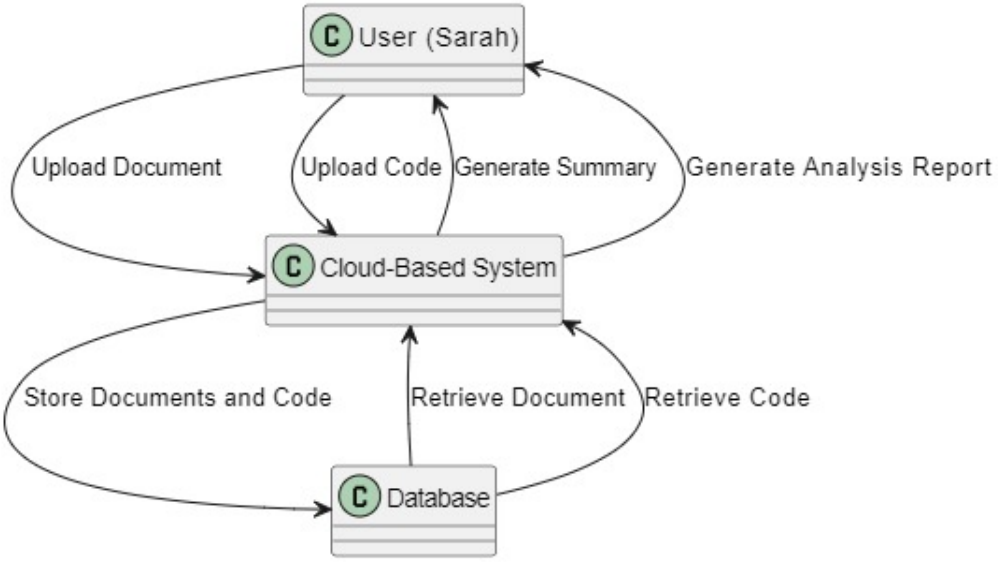

Figure 3.4: Data Flow Diagram (DFD) Level 0

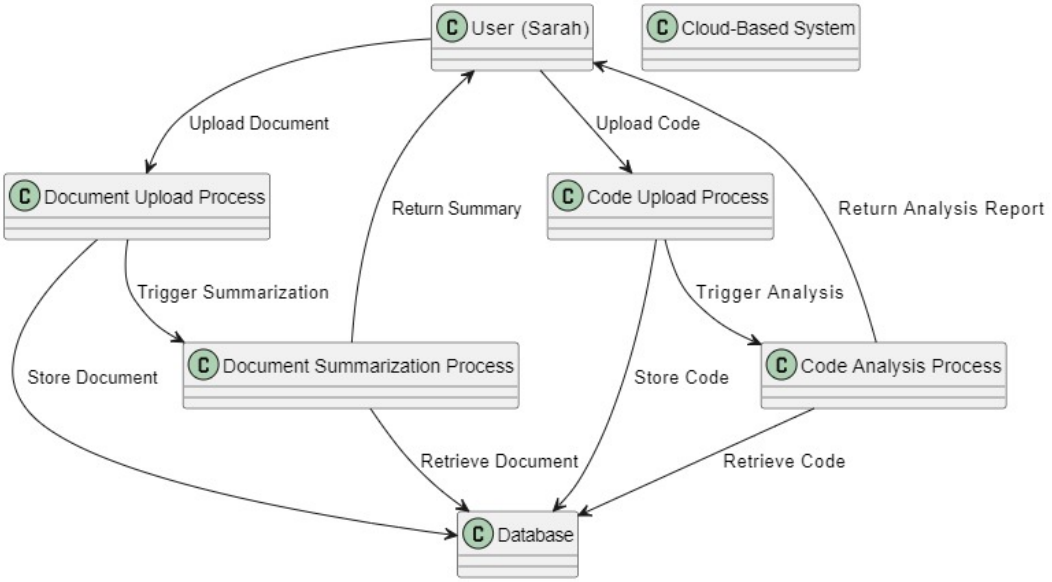

Figure 3.5: Data Flow Diagram (DFD) Level 1

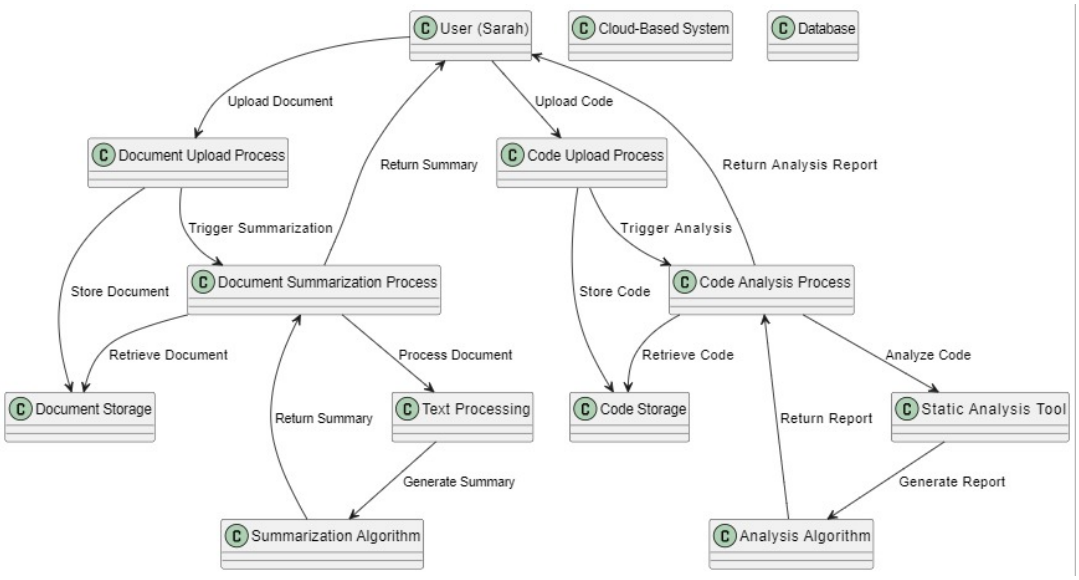

Figure 3.6: Data Flow Diagram (DFD) Level 2

3.3.4 ER Diagram

The ER diagram illustrates the relationships between the various entities in the system. The main entities include:

- **User:** Represents a user who uploads documents or code for analysis.
- **Document:** Contains metadata for uploaded documents (e.g., title, file type, upload date).
- **Summary:** Stores the generated summary of a document or code analysis re-

sult.

- **Code Analysis Report:** Contains details about complexity, bugs, and other code metrics.

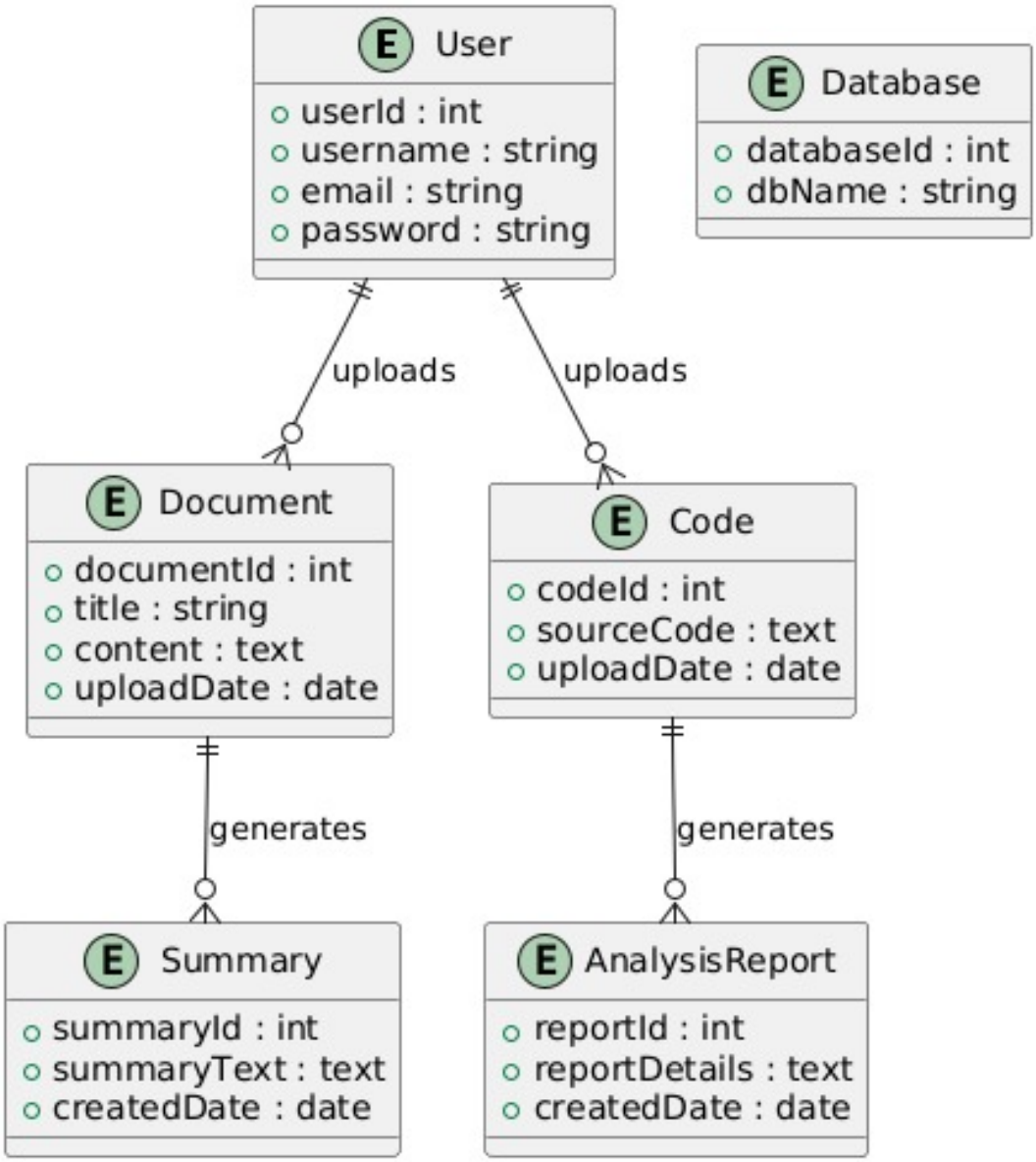

Figure 3.7: ER Diagram

3.3.5 Class Diagram

The Class Diagram shows the structure of the system in terms of classes, their attributes, methods, and relationships (associations).

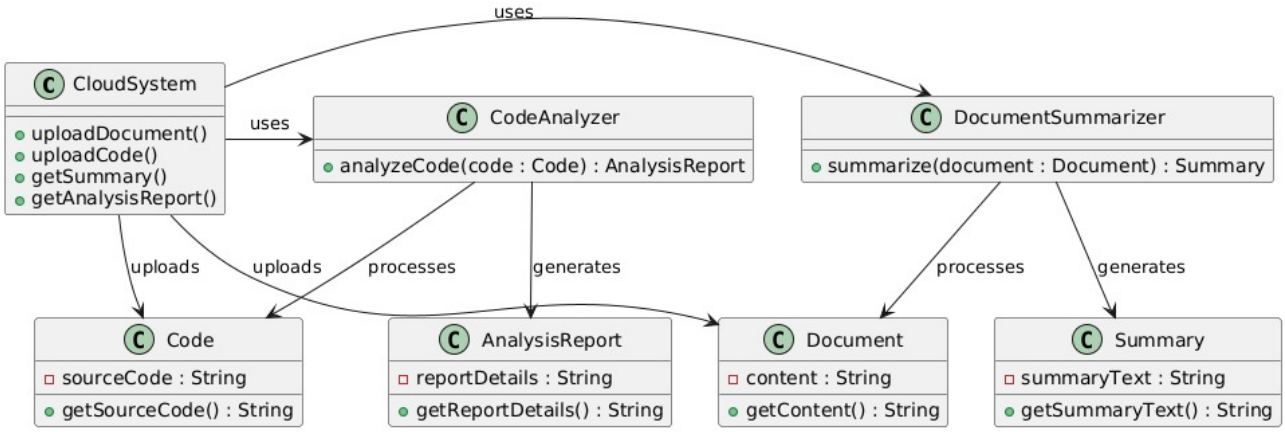

Figure 3.8: Class Diagram

3.3.6 Communication Diagram

The Communication Diagram shows how objects (such as users, the system, and external components) interact with each other through messages.

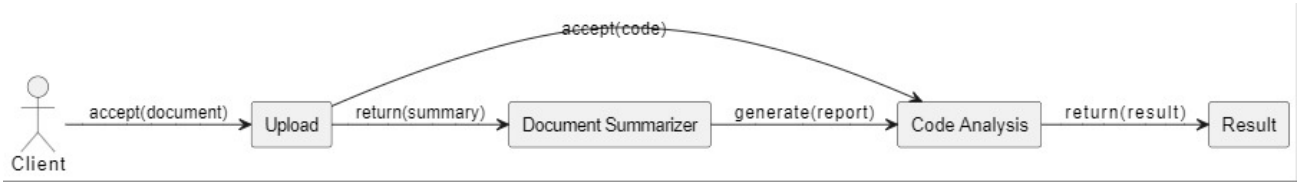

Figure 3.9: Communication Diagram

3.3.7 Sequence Diagram

The sequence diagram demonstrates the communication flow between users and the system for key functionalities, such as uploading a document or code file, processing it using LLMs, and returning the summary or analysis.

- **User Action:** The user initiates a request by uploading a document.
- **System Process:** The system interacts with AWS services (e.g., Lambda, API Gateway) to process the file.

- **LLM Summarization/Code Analysis:** The LLM model or code analysis tools perform the summarization or analysis.
- **Response:** The summarized document or analysis report is returned to the user through the web interface.

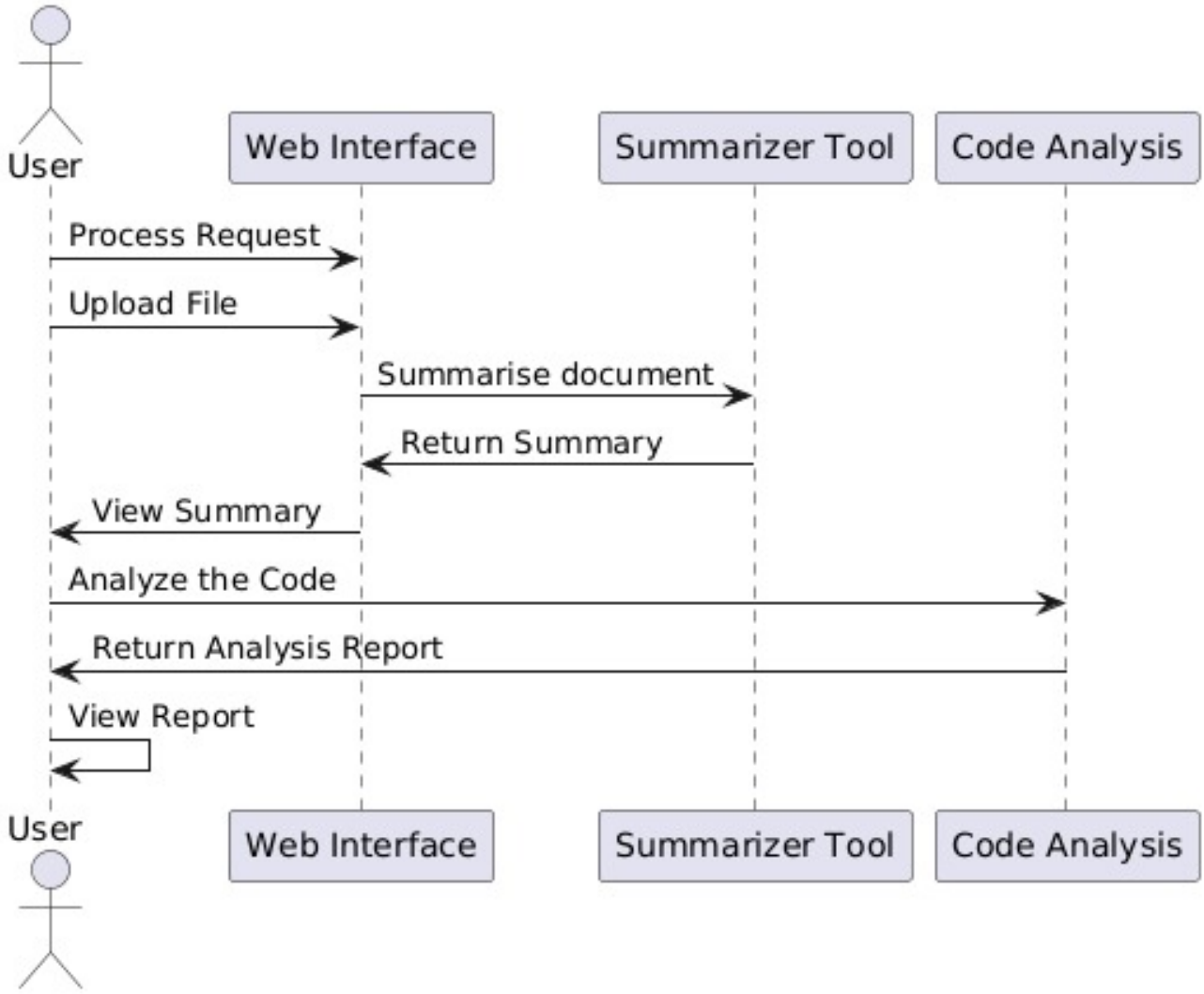

Figure 3.10: Sequence Diagram

3.3.8 Component Diagram

The component diagram provides an overview of the system's structure and shows the relationship between different modules:

- Web Interface: Allows users to upload documents and code files.

- Summarization Module: Receives the input and uses LLMs (GPT-3.5) to process text.
- Code Analysis Module: Utilizes tools to perform code analysis.
- Cloud Infrastructure: Hosts the web interface, summarization engine, and analysis tools.

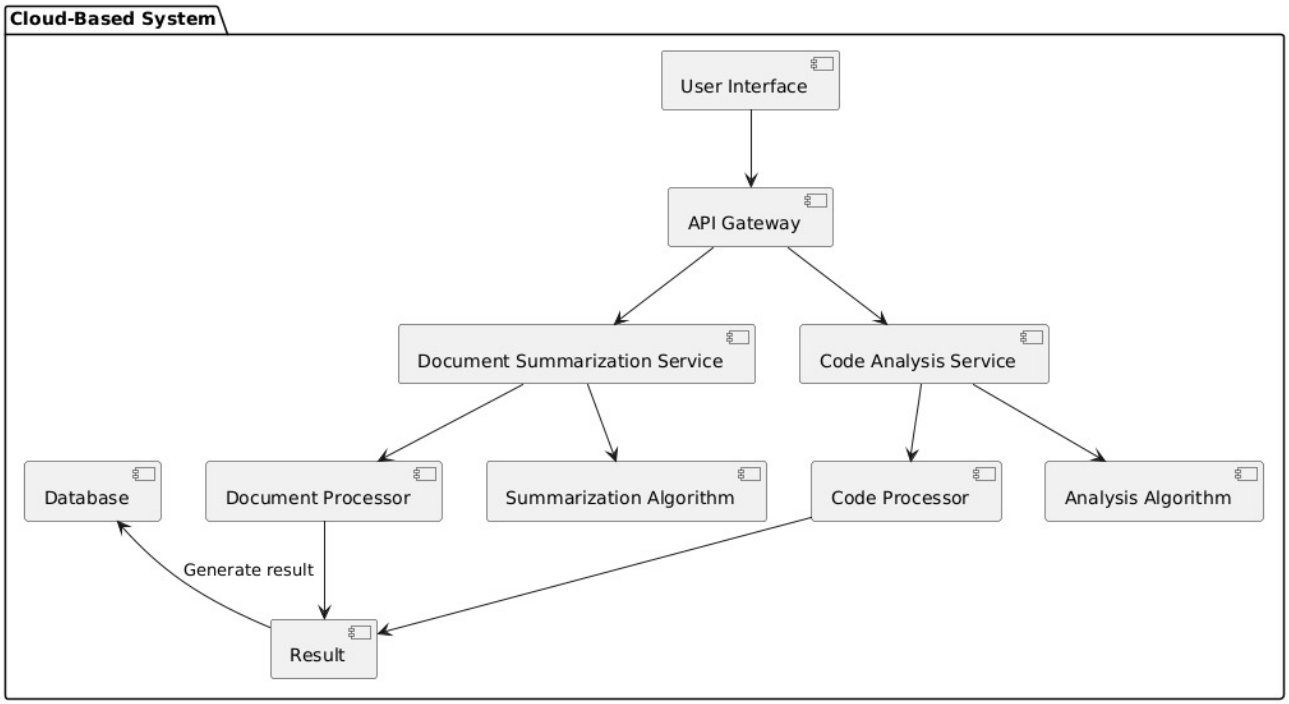

Figure 3.11: Component Diagram

3.3.9 Deployment Diagram

The deployment diagram illustrates how the system is deployed across AWS services:

- Web Server: Hosted on Azure Databricks, which provides the user interface for document and code uploads.
- Lambda Functions: Execute LLM summarization and code analysis in a serverless environment.
- API Gateway: Facilitates communication between the client and backend services.

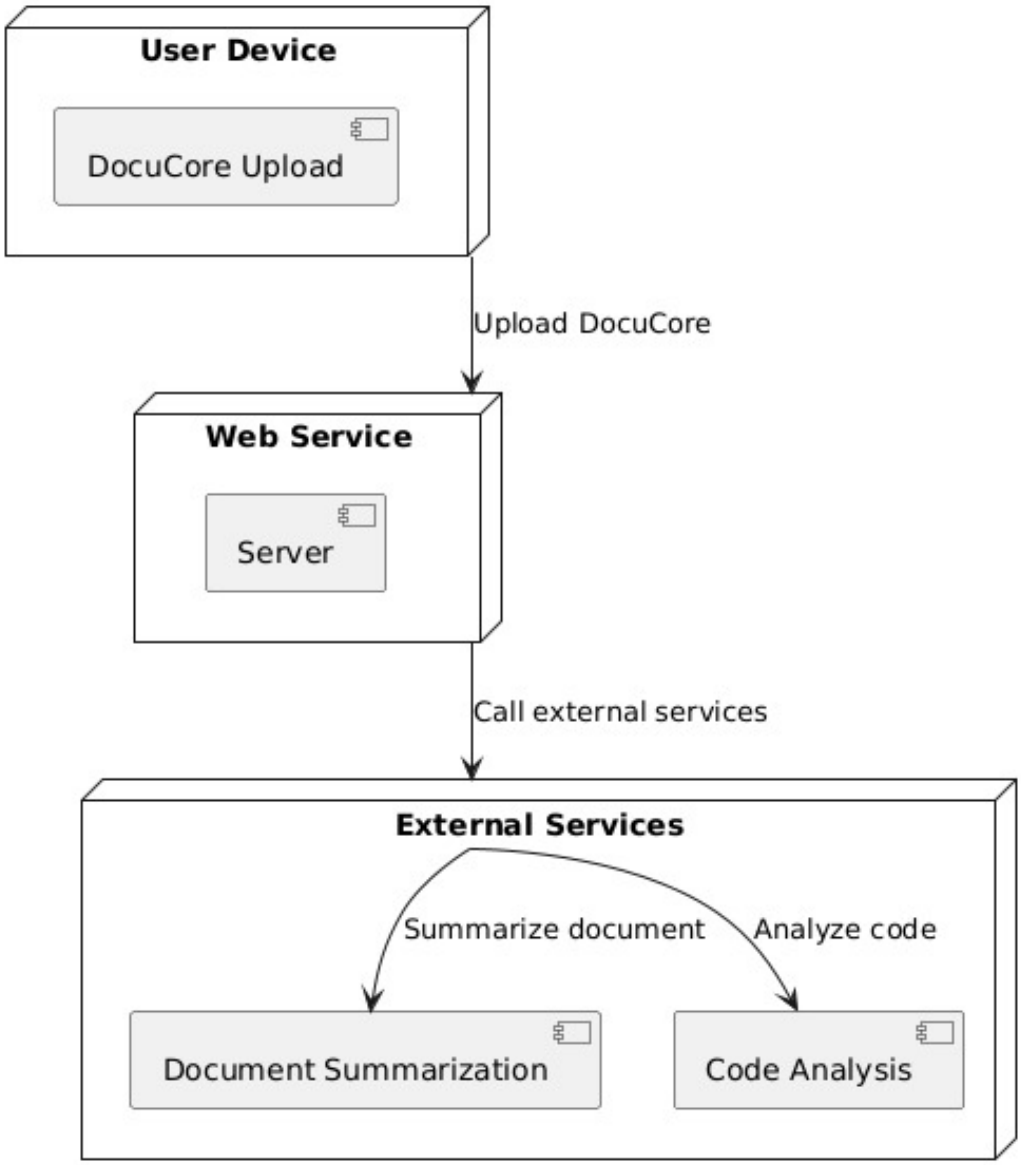

Figure 3.12: Deployment Diagram

3.3.10 Business Process Model

- **Document Summarization Module**
  - Objective: Provide automatic summarization of uploaded documents.
  - Process: User uploads a document via the web interface.Document is sent to the LLM model for summarization.The LLM processes the document and generates a concise summary.
- **Code Analysis Module**

- Objective: Analyze uploaded code for quality, performance, and bugs.
- Process: User uploads code for analysis.The code file is processed by integrated code analysis tools.The system checks for bugs, complexity, and security vulnerabilities.A detailed analysis report is generated.

- **Web-Based User Interface Module**
  - Objective: Enable interaction between the user and the system through a user-friendly interface.
  - Process: Users access the platform via a web browser.The interface allows users to upload documents or code files, view summaries, and analysis reports.Provides access to document management, and report viewing. Notifications or error messages are shown in case of failed uploads or processing.

- **Cloud Integration Module**
  - Objective: Ensure scalability and availability using cloud services.
  - Process: System components are deployed on Azure cloud. User requests are processed in real time by the backend on Azure.The system dynamically scales based on the number of user requests and usage load.

The business process model shows the end-to-end process flow of the system:

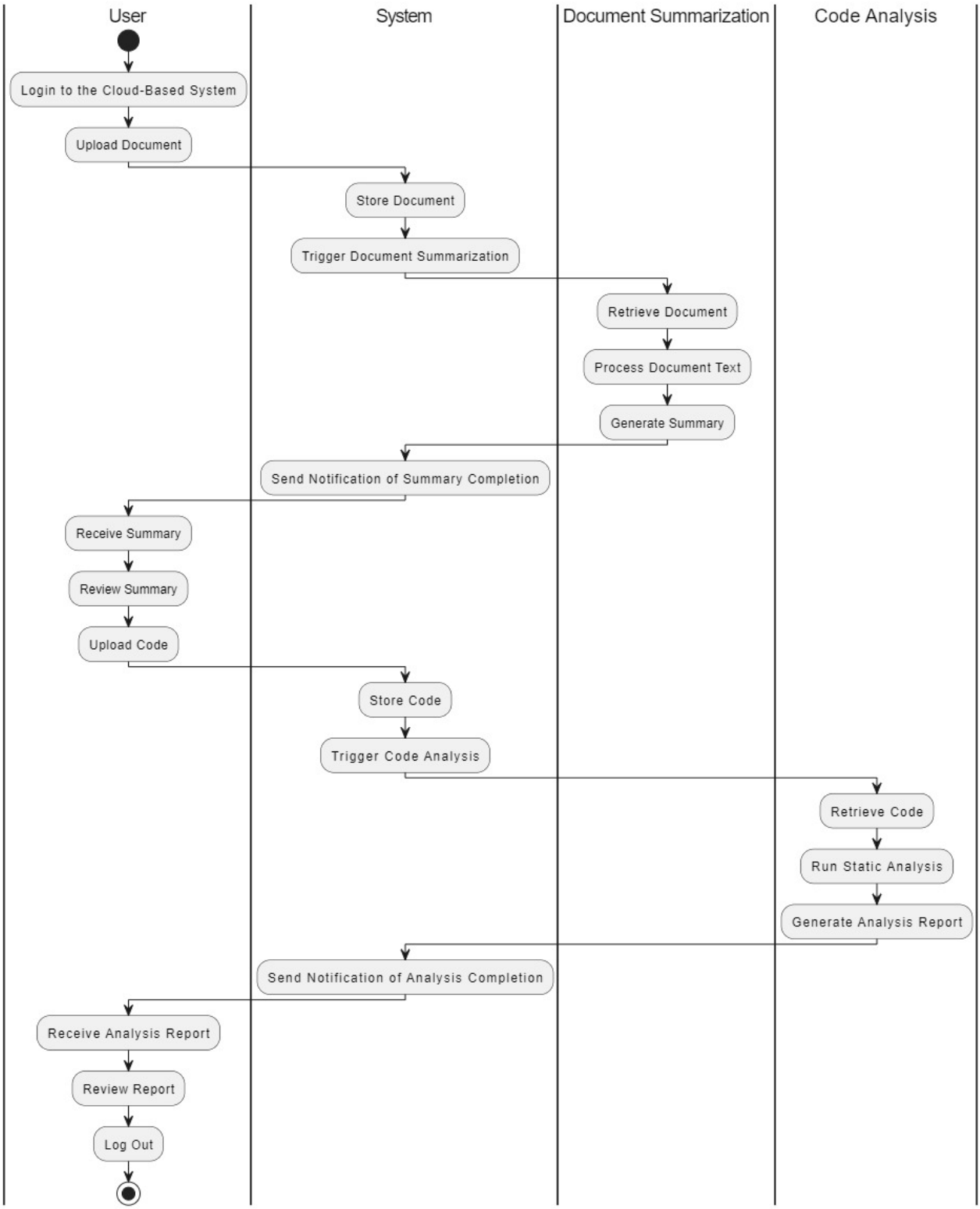

Figure 3.13: Business Process Model

Chapter 4

Methodology and Team

4.1 Introduction to Waterfall Framework

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as an input for the next phase sequentially. Following is a diagrammatic representation of different phases of waterfall model.

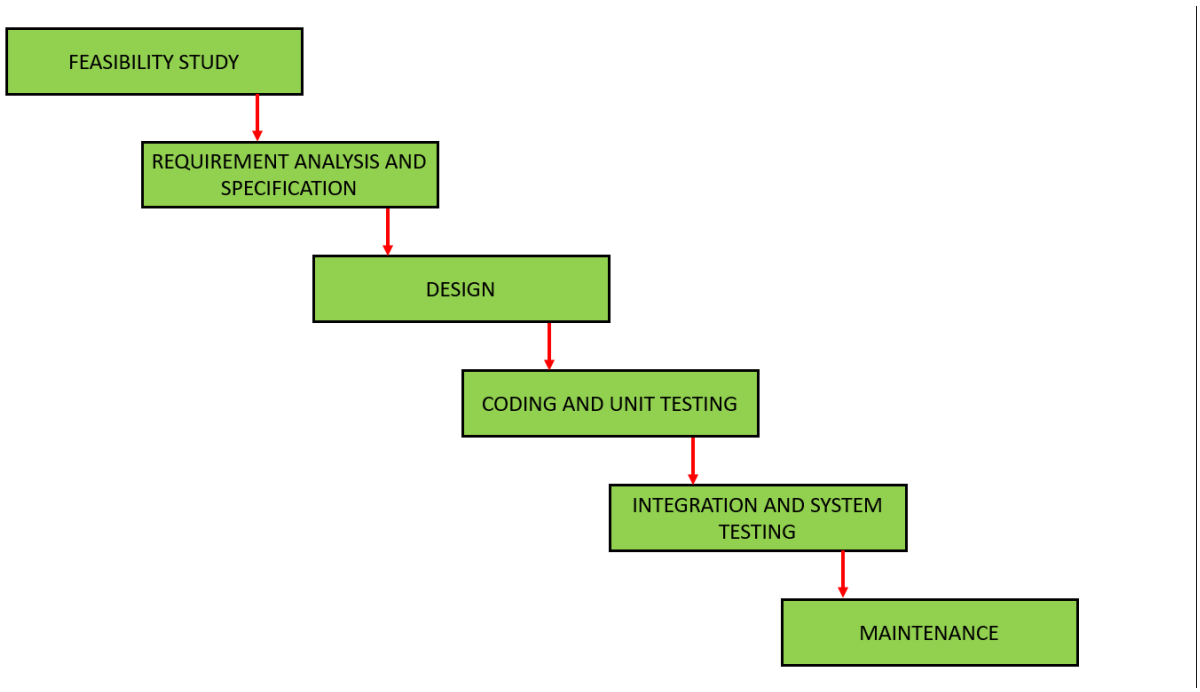

Figure 4.1: WaterFall model

The sequential phases in Waterfall model are-

1. **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

2. **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

3. **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

4. **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

5. **Deployment of system:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

6. **Maintenance:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name ”Waterfall Model”. In this model phases do not overlap.

Waterfall Model Pros & Cons

Advantage The advantage of waterfall development is that it allows for departmen-

talization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one. Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Disadvantage The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

4.2 Team Members, Roles & Responsibilities

The project was completed by a team of three students, with well-defined roles assigned to each member.

- **Garvita Sakhrani (21ESKIT049) — Text Summarisation Module:**
  Developed and integrated the document summarisation module using GPT-based LLMs on Azure Databricks. Handled orchestration of summarisation workflows and integration with backend services.
- **Hiteshi Agrawal (21ESKIT056) — Code-Based Analysis Module:**
  Designed and implemented static code analysis features using tools like SonarQube. Built processing logic for identifying code smells, vulnerabilities, and optimization points.
- **Abhishek Gupta (21ESKIT003) — Cloud Integration Module:**
  Managed Azure Functions, Blob Storage, and deployment pipelines. Configured the Microsoft self-hosted agent and implemented CI/CD workflows using Azure DevOps.

Chapter 5
Centering System Testing

The designed system has been testing through following test parameters.

5.1 Functionality Testing

Functionality testing was conducted to ensure that the system performs all its core features reliably, as defined in the requirement specifications. Each functional module was tested individually and then integrated for end-to-end validation.

- **Document and Code Upload:** Users were able to upload .pdf, .docx, and .txt documents, as well as code files like .py, .java, and .js through the frontend interface. Validation checks for file size (max 50MB) and type were verified.
- **Text Summarisation:** Uploaded documents were processed via Azure Databricks notebooks using GPT-based summarisation. The summarised content was contextually correct and concise.
- **Static Code Analysis:** Code files were analyzed using integrated tools (like SonarQube) for bugs, vulnerabilities, and code smells. Reports included metrics such as cyclomatic complexity, duplication, and critical issue detection.
- **Cloud Integration:** Azure Functions triggered backend summarisation or analysis logic. Azure Blob Storage successfully stored input and output files. The APIs provided expected responses for all tested routes (e.g., `/upload`, `/result`).
- **Error Handling:** Incorrect file formats or failed API calls returned appropriate error messages to the UI. Logs were recorded via Azure Monitor for traceability.

5.2 Performance Testing

Performance testing aimed to assess how the system behaves under various load conditions, including simultaneous user activity and large file processing.

- **Latency Measurement:** The time between file upload and output generation (summary/report) was recorded. Average processing time for a 5-page document was around 12 seconds.
- **Concurrent Upload Simulation:** 10 simultaneous uploads were tested using Postman runner. Azure Functions auto-scaled efficiently to handle the load without failures.
- **Databricks Job Load Handling:** LLM summarisation jobs on Databricks executed within acceptable ranges, even during peak loads. Cluster scaling was tested under batch submissions.
- **Pipeline Performance:** Azure DevOps pipelines with a self-hosted agent completed builds and deployments in under 2 minutes, thanks to caching and local control of dependencies.

5.3 Usability Testing

Usability testing focused on how intuitively users could interact with the system and how well it communicated success, error, or status feedback.

- **User Interface Clarity:** The web UI was tested by both technical and non-technical users. All users were able to complete upload and result-viewing tasks with minimal instruction.
- **Instructional Messaging:** Upload limits, success confirmations, and error alerts were clearly displayed. File type and size errors were properly caught and explained.

- **Design Responsiveness:** The interface adapted well across screen sizes including desktops and tablets. Visual indicators (like loading spinners and status updates) improved overall user confidence.
- **Feedback Incorporation:** Based on peer and faculty suggestions, improvements were made to button labeling and result preview layout.

Chapter 6
Test Execution Summary

Execution Test Summary Report provides a consolidated view of the entire testing lifecycle — from the initiation of test planning to the completion of test execution. While the Test Plan outlines the strategy and scope of testing at the beginning of the project, the Test Execution Summary Report captures the actual test outcomes at the end of the testing phase.

This document helps stakeholders understand how thoroughly the system has been validated and whether it meets the quality expectations. The summary below includes the essential metrics tracked during the testing of the DocuCore platform. The Test Summary Report contents are :

1. Test Case ID generated
2. Total number of resources consumed
3. Passed Test Cases
4. Failed Test Cases
5. Status of Test Cases

S.No	Test Case ID	Test Case Description	Test Case Status	Resources Consumed
1	TC-FUNC-01	Upload document and validate size/type	Passed	VSCode, Azure Blob
2	TC-FUNC-04	Execute LLM-based document summarisation	Passed	Azure Databricks, Function App
3	TC-FUNC-06	Run code analysis using static scanner	Passed	CI/CD Pipeline
4	TC-UI-02	Display result summaries and reports	Passed	Browser, Web UI, Azure Blob
5	TC-PERF-03	Measure response time for concurrent uploads	Passed	Azure Monitor
6	TC-FUNC-09	Handle unsupported file types gracefully	Failed (resolved and passed after fix)	Function Logs, Error Handler

Table 6.1: Test Execution Summary Table

Chapter 7
Project Screenshots

The following screenshots showcase the core features, backend infrastructure, and deployment mechanisms of the project *DocuCore: Text & Code Summariser in Cloud*. Each screenshot represents a critical part of the system, from user interface to backend automation using Azure, Databricks, CI/CD pipelines, and custom deployment agents.

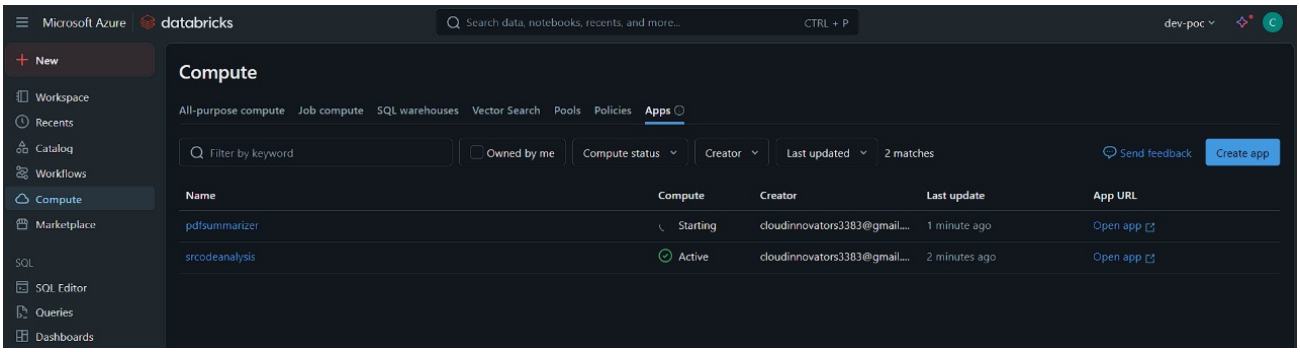

Figure 7.1: Home Interface Displaying Access to Summarisation and Code Analysis Modules

The above image shows the Databricks Compute interface with two deployed apps — `pdfsummarizer` and `srccodeanalysis`. These allow users to interact with summarisation and code analysis workflows, deployed modularly for maintainability and scalability.

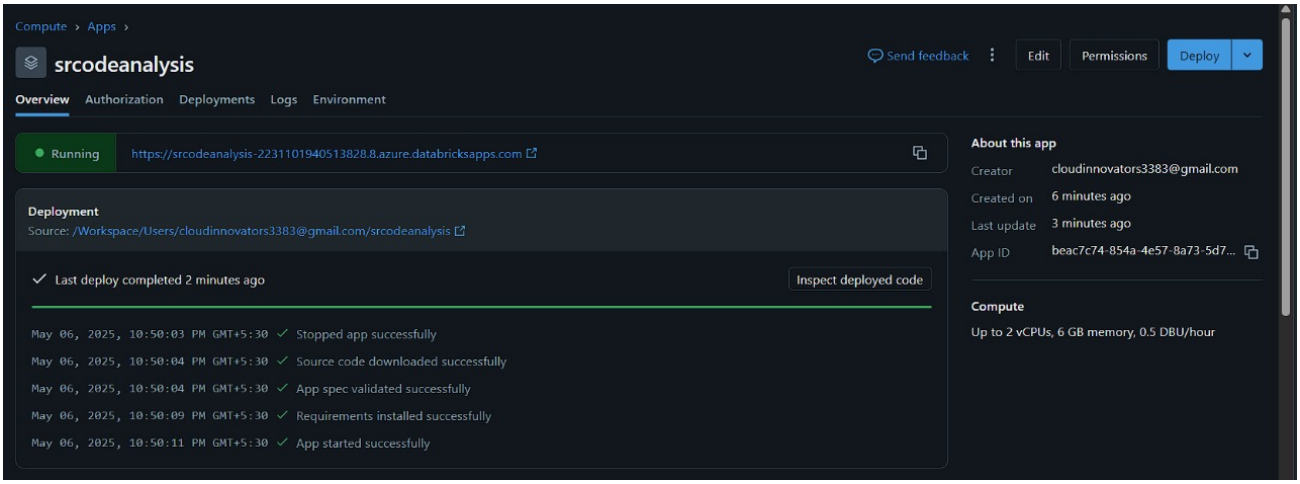

Figure 7.2: Deployment Interface

This deployment log verifies that the source code analysis app was deployed successfully. The system stopped previous versions, downloaded the source, set up the environment, and started the app using Azure Databricks automation.

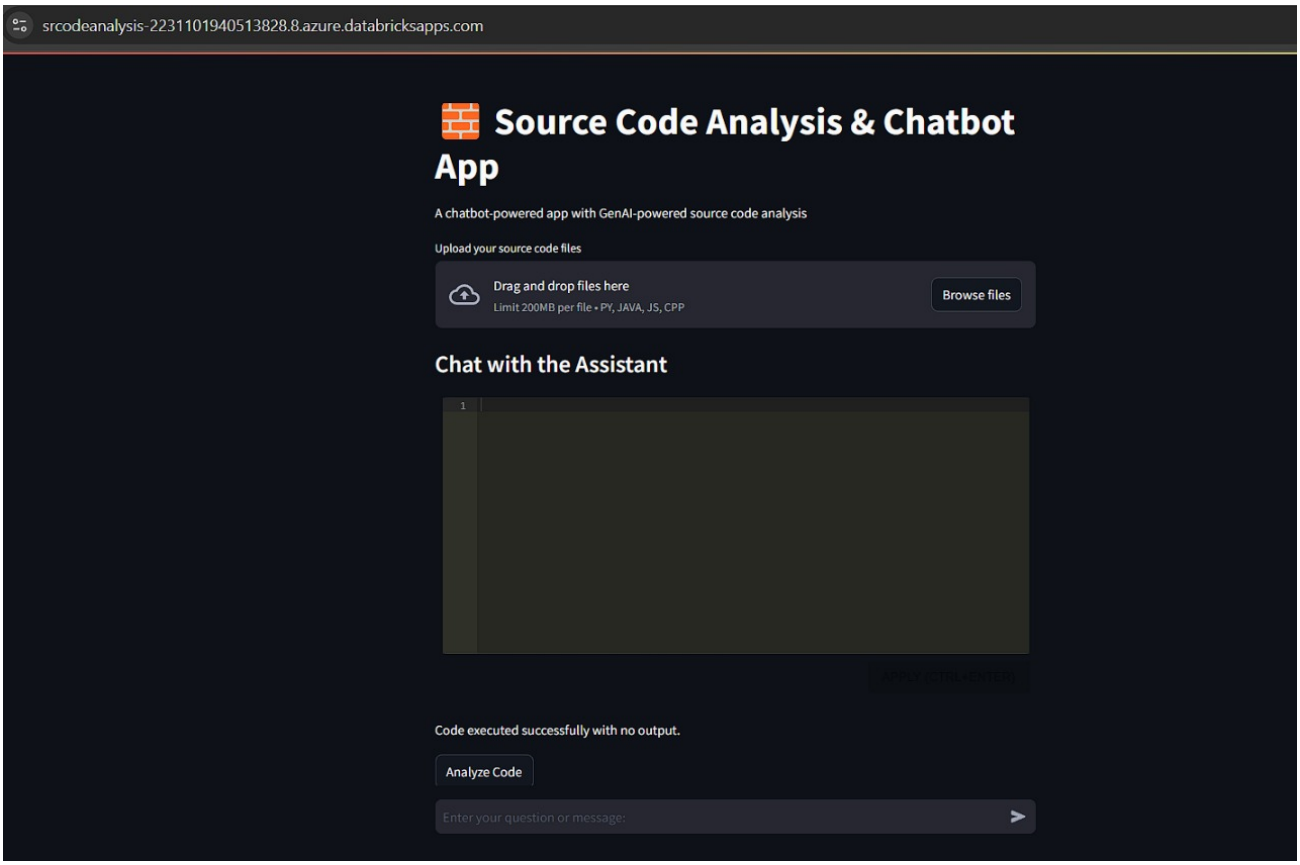

Figure 7.3: Source Code Analysis Dashboard

The chatbot interface allows users to upload files like `.py`, `.java`, and `.cpp` and interact with the app to perform GenAI-powered code analysis using natural language queries.

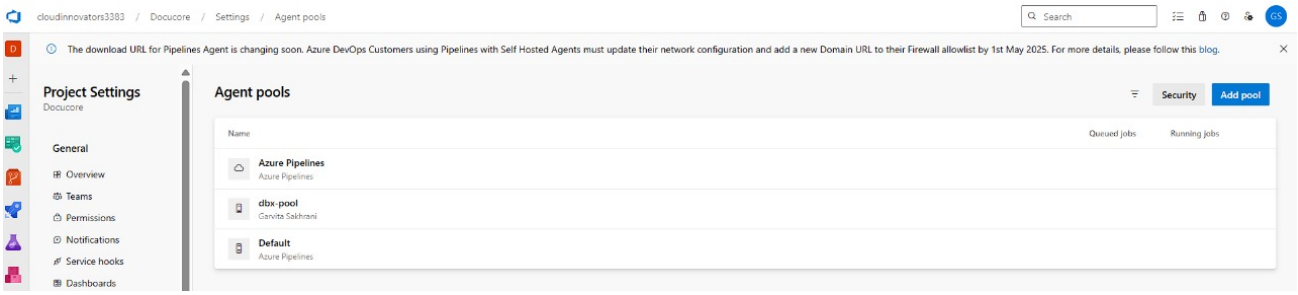

Figure 7.4: Agent Pools

This screenshot displays the DevOps agent pools where a custom pool `dbx-pool` is configured. This allows CI/CD pipelines to run using a Microsoft Self-Hosted Agent hosted on an Azure VM.

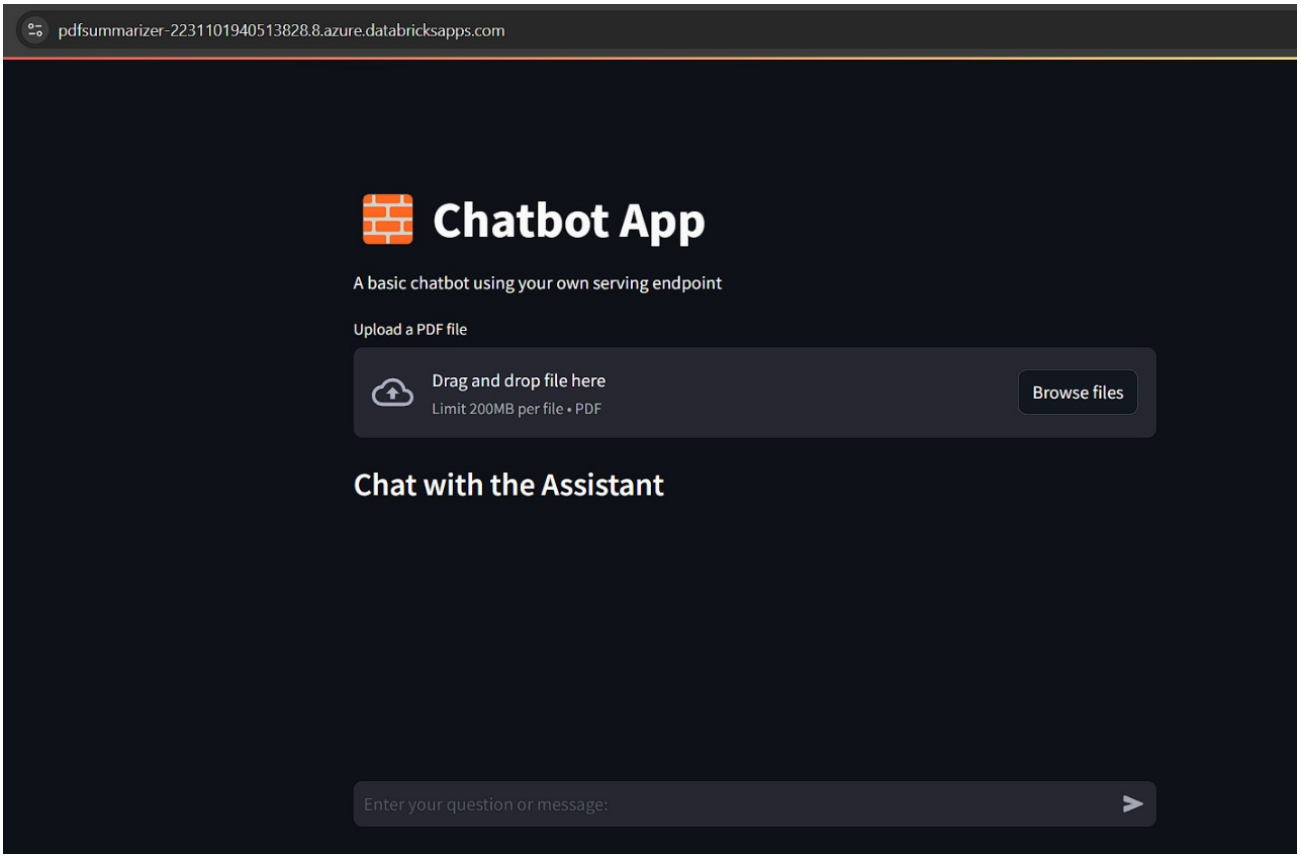

Figure 7.5: Chatbot App Dashboard

This is the PDF summarisation interface. It allows users to upload technical documents (PDF) and chat with an assistant to get meaningful summaries powered by LLMs running in Azure Databricks.

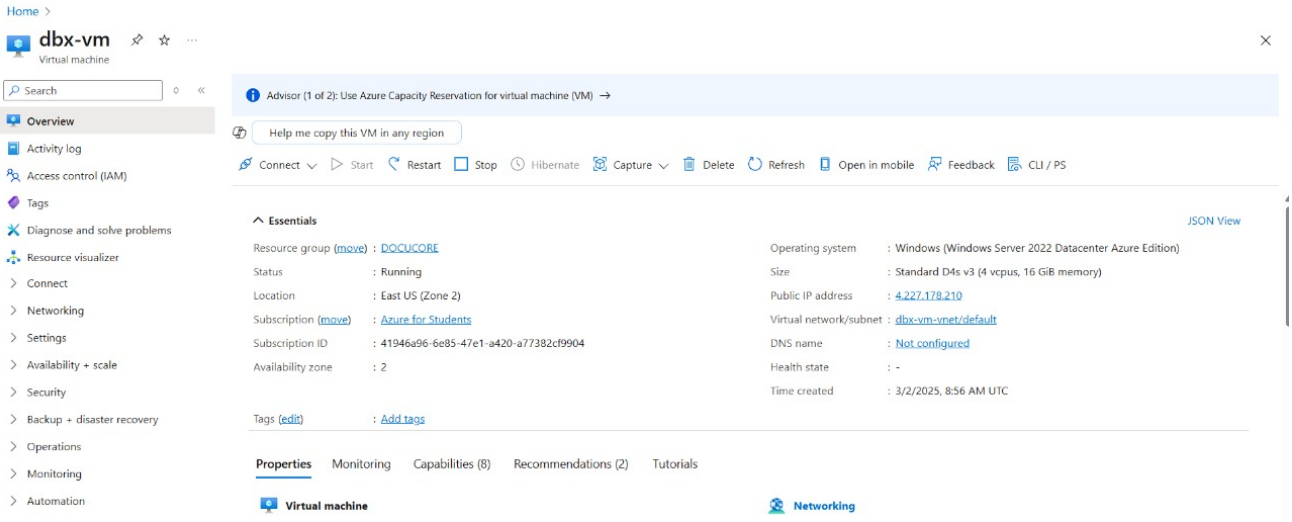

Figure 7.6: Databricks V.M. Configuration

This Azure VM hosts the self-hosted agent used by Azure DevOps to run pipeline jobs. The VM is configured with 4 vCPUs and 16 GB RAM and acts as the core deployment runner for all CI/CD tasks.

Chapter 8
Project Summary and Conclusions

8.1 Project Summary

DocuCore: Text & Code Summariser in Cloud is a cloud-based platform designed to address the growing complexity of managing large codebases and technical documents in modern software development. In today's fast-paced environment, developers face challenges in understanding lengthy documentation and ensuring code quality, often leading to productivity bottlenecks and errors.
To solve this, the project integrates intelligent automation and cloud computing into a unified solution that streamlines both document summarisation and code analysis. Unlike existing tools that work independently, DocuCore offers a seamless pipeline where text and code are processed, analyzed, and stored in real-time.
The system leverages the following key technologies:

- **Microsoft Azure:** Used for hosting backend services, file storage (Azure Blob), serverless functions, and monitoring.
- **Azure Databricks:** Executes LLM-based summarisation workflows and code analysis logic on Spark-enabled clusters.
- **Git and Azure Repos:** Ensures source control and collaborative development.
- **CI/CD Pipelines via Azure DevOps:** Automates testing and deployment using YAML-based pipelines.
- **Microsoft Self-Hosted Agent (SHA):** Provides flexibility, custom environment configuration, and faster builds in the DevOps pipeline.
- **Visual Studio Code (VSCode):** Primary IDE used for code development, debugging, and extension integration.

The project follows a clear logical flow: **Upload → Process → Store → Display.** Users upload text documents or code files via a user-friendly web interface. These inputs are automatically processed—text via GPT-based summarisation and code via static analysis tools like SonarQube. The results are stored on Azure Blob Storage and displayed back to users.

DocuCore bridges the gap between isolated tools and creates a centralized automation platform that increases efficiency, improves accuracy, and enhances developer productivity.

8.2 Conclusion

DocuCore successfully demonstrates how cloud-native technologies and AI-driven automation can be integrated into a single unified platform to simplify the traditionally manual tasks of document summarisation and code review.

By automating these processes through scalable pipelines on Azure and Databricks, and orchestrating them using CI/CD mechanisms via Azure DevOps, the project proves that routine and complex tasks can be handled more effectively without constant developer intervention.

The inclusion of a Microsoft Self-Hosted Agent allows better customization and control over the CI/CD environment, while services like Azure Monitor and Application Insights ensure system reliability, traceability, and observability.

The simplicity of the web interface, the depth of backend automation, and the seamless integration of tools reflect a strong architectural foundation and modern engineering practice. Most importantly, the solution addresses a clear need in the development community — providing a faster, smarter way to understand and act on large volumes of technical content.

DocuCore is not just a student project, but a working prototype of a system that could evolve into an enterprise-ready tool for development teams. It sets a solid foundation for future enhancements such as multilingual support, real-time collaboration, authentication, and integration with AI-driven recommendation engines.

Chapter 9
Future Scope

The current implementation of **DocuCore: Text & Code Summariser in Cloud**lays a solid foundation for scalable document summarisation and static code analysis using Microsoft Azure and Databricks. However, there are several directions in which the system can be further enhanced and optimized in future iterations:

- **Rollback Deployment Mechanism:**
  Implement automated rollback procedures within the Azure DevOps CI/CD pipeline to ensure that if any deployment fails or causes instability, the system automatically reverts to the previous stable version without manual intervention.

- **Multi-Language Support for Code and Documents:**
  Expand the system to support additional programming languages like C++, C#, and JavaScript, and document formats such as Markdown (.md), LaTeX (.tex), and XML. This will make the system more versatile and useful across a wider developer audience.

- **Advanced Network Security Practices:**
  Strengthen platform security using Azure best practices, such as role-based access control (RBAC), API gateway throttling, HTTPS-only policies, and firewall configurations. Future enhancements could also include end-to-end encryption for file uploads and access tokens.

- **Authentication and Access Control:**
  Introduce user authentication using Azure Active Directory or OAuth to enable personalized access, history tracking, and usage analytics per user. Access levels (e.g., admin, contributor, viewer) could be defined for multi-user scenarios.

- **Mobile Application Integration:**

Develop a lightweight mobile application or progressive web app (PWA) that allows users to upload documents or review summarisation reports on the go, enhancing accessibility and real-time interaction.

- **AI-Powered Recommendations:**
  Integrate GPT-based recommendation systems that suggest improvements for code readability, performance optimization, or documentation clarity based on static analysis results.

- **Real-Time Collaboration and Sharing:**
  Enable shared workspaces where multiple users can collaborate on document/-code analysis sessions, comment on reports, or make collective decisions.

- **Versioning and Report History:**
  Store and manage version histories of uploaded files and their associated analysis or summaries. Users will be able to compare versions and track improvements or regressions over time.

- **Notification System:**
  Implement email or in-app notifications to alert users when summarisation/-analysis jobs are complete or when deployment changes affect their data.

- **Integration with Developer Tools:**
  Integrate the summarisation and code analysis services as plugins/extensions within popular IDEs like VSCode, IntelliJ, or GitHub Actions.

These future enhancements will help transform DocuCore into a more robust, intelligent, and user-friendly enterprise-grade platform, ready to handle complex workflows, larger teams, and mission-critical documentation or source code at scale.

References

[1] OpenAI, *GPT-3: Language Models are Few-Shot Learners*, OpenAI, 2020. [Online]. Available: https://platform.openai.com/docs. [Accessed: Apr. 21, 2025].

[2] Microsoft, *Microsoft Azure Documentation*, Microsoft, 2025. [Online]. Available: https://learn.microsoft.com/azure. [Accessed: Apr. 24, 2025].

[3] Microsoft, *Azure DevOps Documentation*, Microsoft, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops. [Accessed: Apr. 22, 2025].

[4] Chroma, *ChromaDB: The Vector Database for AI-First Applications*, Chroma, 2025. [Online]. Available: https://www.trychroma.com. [Accessed: Apr. 22, 2025].

[5] Databricks, *Databricks Unified Analytics Platform*, Databricks, 2025. [Online]. Available: https://azure.microsoft.com/en-in/products/databricks. [Accessed: Apr. 25, 2025].