**صورة تحتوي على شعار, رمز, علامة تجارية, دائرة

تم إنشاء الوصف تلقائياً**

**Faculty of Computer and Information Technology**

**Department of Computer Engineering**

**CPE 591: Graduation Project II Report**

**[AI-Based interactive digital content application: search, summarize, save, translate, integrate, chat and present]**

**Noor Saleh Sharkawi 143673**

**Aya Amjad Mahmoud 145227**

**Supervised By**

**Dr.Lo'ai Tawalbah**

**Winter 2024-2025**

## Table of Contents

[**Acknowledgment**](#page5)**……………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

**Introduction…………………………………………………………………………………………..**[**5**](#page5)

**Professional Practice Constraints…………………………………..………………………………..**[**5**](#page5)

**System Architecture and Design………………………………………………………………..**[**5**](#page5)

**Software Implementation…………………………………………………………………………..**[**5**](#page5)

**Project Planning and Task Definition……………………………………………………………..**[**5**](#page5)

**Literature Review……………………………………………………………………………………..**[**5**](#page5)

**Preliminary Design ………………………………………………………………………………..**[**5**](#page5)

**Detailed System Design……………………………………………………………………………..**[**5**](#page5)

[**Conclusion**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

**Recommendations…………………………………………………………………………………..**[**5**](#page5)

[**List**](#page5) **of References …………………………………………………………………………………..**[**5**](#page5)

[**Appendices**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Meeting**](#page5) **Minutes ……………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

[**Abstract**](#page5)**……………………………………………………………………………………………..**[**5**](#page5)

**Table of Figures**

Figure 1: System Architecture………………………………………………………

Figure 2: Chatbot Component ………………………………..……………………..

Figure 3: Summarization System………………………………………………….

Figure 4 : Translation component…………………………………………………

Figure 5:Notes Component…………………………………………………………..

Figure 6 :To Do List…………………………………………………………………

Figure 7 :Add File …………………………………………………………………

Figure 8 : Bookmark & Continue Reading………………………………………….

**List of Tables**

Database schemas………………………………………………………………………………

* Figure 1:Users
* Figure 2:userFiles
* Figure 3:AllUsersFiles

## 2. Acknowledgment

We would like to express our sincere appreciation to Lo'ai Tawalbah, our supervisor, with deepest gratitude for his useful guidance, continuous encouragement, and all the feedback he has provided us throughout this project.

We would like to extend our appreciation to the Jordan University of Science and Technology (JUST) and the Computer Engineering Department (CPE) for affording us the resources and information that enabled this project to succeed.

A heartfelt appreciation to our friends and families for their continued encouragement and patience over this period. They have been a source of constant motivation.

Lastly, we would like to thank all those who, either directly or indirectly, assisted in ensuring the success of this project. Your guidance and support have been most vital in bringing our dream to fruition.

## 3. Abstract

In today's digital era, effective content management and accessibility are essential for students, researchers, and working professionals. This project introduces an AI-based digital content application focusing on file management and document interaction, not just reading files. The app incorporates a smart chatbot and summarization through the Gemini API, which makes working with documents easier. The frontend is created with Flutter for smooth user experience, and the backend is created with Python and Flask for consistent performance.

It contains searchable PDFs, which allow users to search for key information with highlighted results for the search. Translation is also supported through a simple widget such as Google Translate. In terms of productivity, users are able to take notes and to-do lists, all synced using Firebase for real-time data management.

Security and user authentication are handled using Firebase Authentication, and Firestore is utilized as the primary database for storing user-generated content. Bookmarks and a 'Continue Reading' feature improve the user experience based on content consumption.

This project aims to provide an effective and intelligent digital content organizational solution for students and working professionals alike. Through the power of AI and cloud services, the system offers a secure yet user-friendly platform that optimizes document management, boosts productivity, and eases information retrieval.

## 4. Introduction

### 4.1 Statement of the Problem

In today's digital era, it is difficult to work with a lot of documents. The majority of file readers show content but do not support productivity. The users spend time looking for crucial information, condensing long text, or organizing notes. Modern tools also don't offer intelligent AI support for summarizing, or document organization in a convenient way all together.

### 4.2 Significance of the Project

This application is designed to make it much easier and quicker to work with documents. It uses AI in a way that enables you to quickly summarize lengthy texts and find important information with search. Documents are securely stored on the cloud using Firebase, so you can view them on any device. In contrast to a typical e-reader, it also comes equipped with useful productivity functions like to-do lists, bookmarks, and a functionality remembering where you stopped reading. Students will benefit from the functionality of this app for research work, and it can be useful to professionals so that they may ensure better handling of working documents. All these smart features together save time and open digital content for everyone.

### 4.3 Goals

The overall purpose of this project is to develop an AI-Based Interactive Digital Content Application to streamline document processing and content reading. The specific goals are:

1.AI Summaries : Google's Gemini AI is being used in the application to create easy-to-read forms of long documents.

2.Smart Chatbot : a chatbot with assistance from an AI helper ( Google's Gemini AI)helps to answer queries.

3.Fast PDF Search :search words with highlighted results in PDFs and navigate throw them.

4. Translation screen translate text instantly to other languages.

5. Cloud Sync save files, bookmarks, last position to continue reading, notes, and to-do lists in Firebase, and view them on any device.

6. Secure Login & Storage your information is secure with Firebase's cloud storage and authentication.

7. Reading convenience the app remembers where you left off and lets you bookmark pages of significance.

8. Easy to use :create a clean, minimalistic interface developed with Flutter for easy navigation.

### 4.4 Contemporary Issues

By solving these common problems, our app makes working with digital documents easier for professionals and students. It saves time and helps people stay organized.

1. Too Much Information to Process

Today, people have to read so many documents that it is hard to know the important information. Our app addresses this by using AI to create short, understandable summaries. This enables users to understand documents faster without needing to read all the material.

2. Obtaining Information in a Timely Fashion

Most document programs do not have good search capability, and the user wastes much time searching for what they are looking for. Our program makes searching easier because it highlights the keywords in PDFs. If you perform a search, you will know where exactly it is located in the document.

3. Keeping You Organized

Most document readers do not have note-taking or to-do list capabilities. This means that people have to use multiple apps for different purposes. Our app has built-in note-taking, bookmarks, and task lists so that everything you need is in one place.

4. Smart AI Help When You Need It

Most document apps don't use AI to help the user. Our app has a handy chatbot that can answer your questions about your documents. It uses Google's Gemini AI technology to give you smart, helpful answers.

### 4.5 Impact on Society

1. Saves Time for Students and Researchers

AI summaries and search within the app allow users to find critical information within a limited time. Instead of going through full documents, they can read the key points.

2. Ideal for Everyone

Thanks to translation software and easy search features, the app is suitable for people with different languages or requirements.

3. Keeps You Organized

In-built notes, to-do lists, and bookmarks spare users the need for multiple apps. All these items combine and function within one app.

4.Up-to-date Digital Tools

The app takes advantage of AI and cloud technology to serve the learning and working needs of today. It helps users work in smarter, more digital ways.

5.Makes Information More Understandable

### 4.6 Initial Constraints

The project was exposed to some early constraints that influenced the design and implementation:

1.Limitations of AI API : The Google Gemini API can have rates or restricted access, which affects performance.

2.Cloud Storage Cost : Firebase storage and Firestore database are costly since data usage increases.

3.Computational Complexity :AI summarization and chatbot responses must be processed in an efficient way to maintain real-time performance.

4.Compatibility Across Platforms: Seamless integration of Flutter (front-end) and Flask (back-end) created a cautious necessity for optimization.

5.Challenges in User Interface Design: Optimization for feature richness combined with UI simplicity presented the principal design challenge.

The above constraints, however, get neutralized by employing strategic planning combined with optimization principles to provide user-friendly and fluid usability.

## 5. Professional Practice Constraints

The design and development of our application adhere to several professional practice constraints, ensuring it meets engineering standards, societal values, and ethical principles. Below is an analysis of the key constraints applicable to the project:

### 5.1. Manufacturability Constraints

While the application is a software product, constraints related to manufacturability are applicable in

terms of code structure, usability, and deployment. The design follows industry standards such as:

· Code Modularity: Ensures ease of maintenance and future scalability

· Adherence to Flutter and Firebase Best Practices: To ensure compatibility and efficient integration

with the chosen tools and frameworks.

· Ease of Use: The user interface prioritizes intuitive navigation, with straightforward access to features

like note-taking, to-do lists, file uploads, and AI-powered tools.

### 5.2. Economic Constraints

The project is designed to be cost-effective for both developers and end-users:

· Development Costs: Firebase offers a free tire, allowing initial development without incurring

significant costs.

· Open-source tools (e.g., Flutter, Flask) are used to minimize expenses.

· Manpower resources: The project team is composed of students, necessitating efficient

allocation of time and effort.

· Time constraints: The project is managed within a limited academic timeframe, requiring

prioritization of features and milestones through effective project management.

### 5.3. Sustainability Constraints

The application incorporates sustainable practices:

· Firebase’s cloud-based infrastructure ensures long-term data storage without the need for physical

resources.

· The app is lightweight, reducing energy consumption during use and ensuring compatibility

with a wide range of devices (Android).

### 5.4. Environmental Constraints

Reduction of paper use by offering digital tools for file summarization, note-taking, and task

management, the app promotes a shift from paper-based workflows to digital solutions,

contributing to environmental conservation.

### 5.5. Health and Safety Constraints

User Data Protection:

· Firebase Authentication ensures secure access to the application, protecting user data from

unauthorized access.

· Firestore rules are implemented to safeguard sensitive information, such as uploaded files and

personal notes.

### 5.6. Ethical Standards Constraints

Respecting user privacy with no unauthorized data collection or sharing, providing transparency in

how the AI summarization and chatbot features function and a voiding any features or functionalities that

could be used for malicious purposes.

### 5.7. Social Values Constraints

The application aligns with societal values to ensure that it meets ethical, cultural, and social

expectations:

· Ethical Usage: The app avoids any features that could be exploited for unethical purposes, such as

special information spread.

· Inclusivity and Accessibility: Designed to cater to a diverse user base, including individual casual

readers and academic users.

### 5.8. Political Constraints

Data security and privacy, particularly under GDPR (General Data Protection Regulation) and similar frameworks. In addition, responsible use of AI and cloud-based technologies, adhering to government and industry standards.

## 6.System Architecture and Design

[Show the system architecture with major components then give details of the design and implementation details for each part.]

****The Figure 1:The system architecture with major components

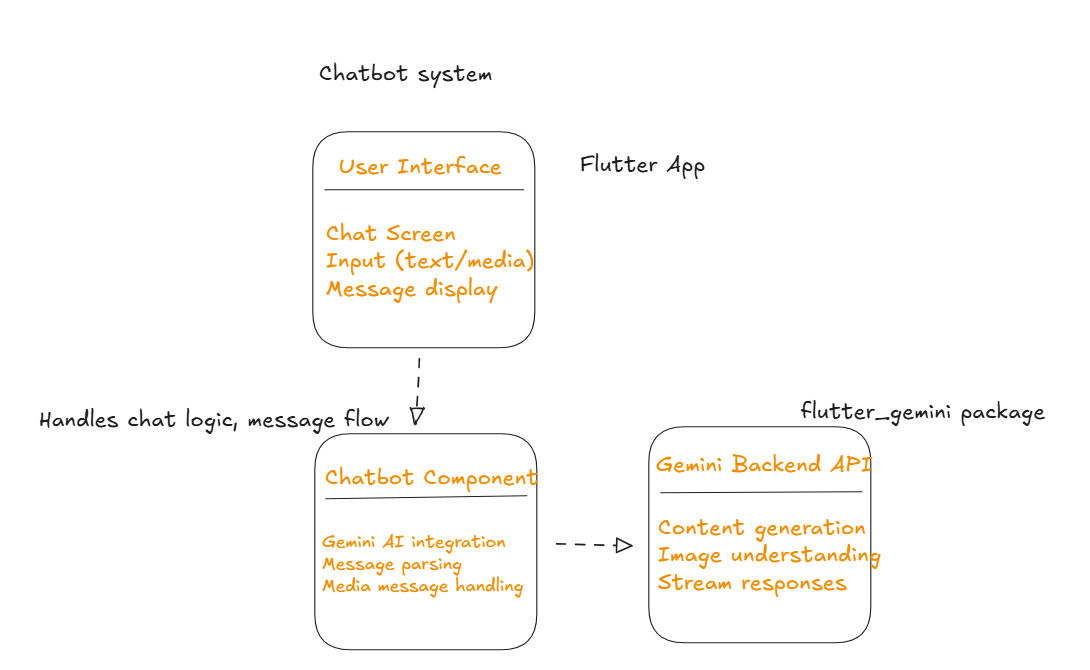
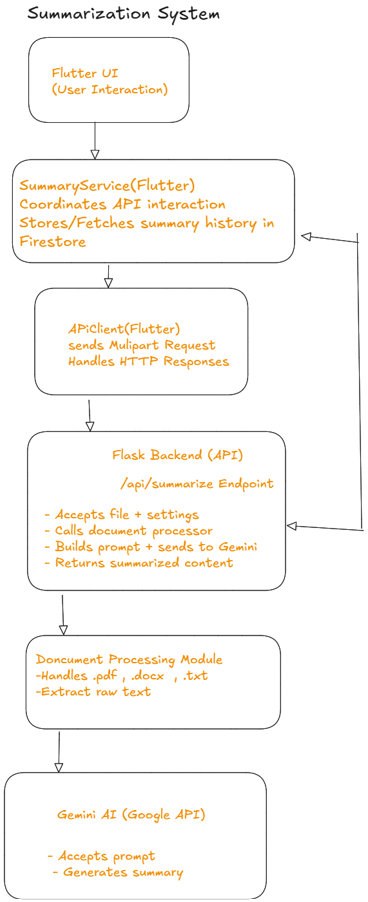
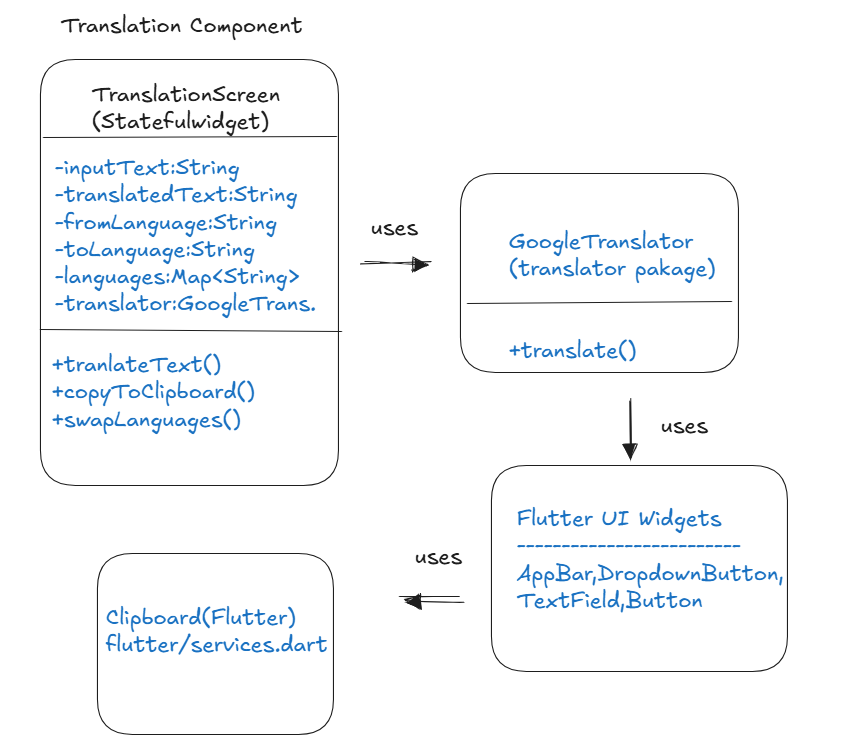
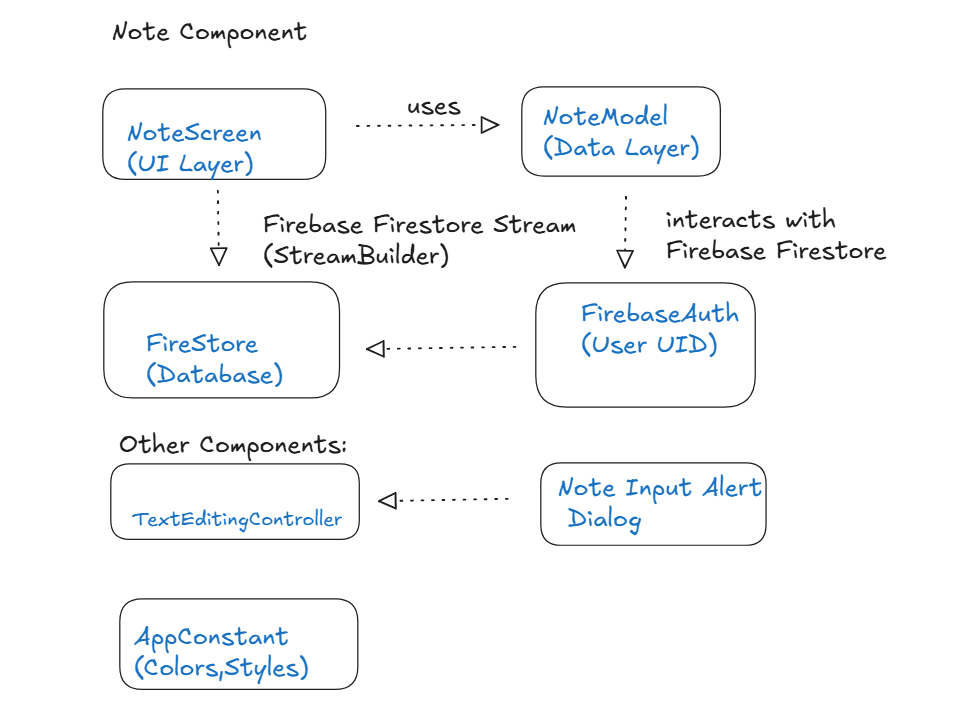
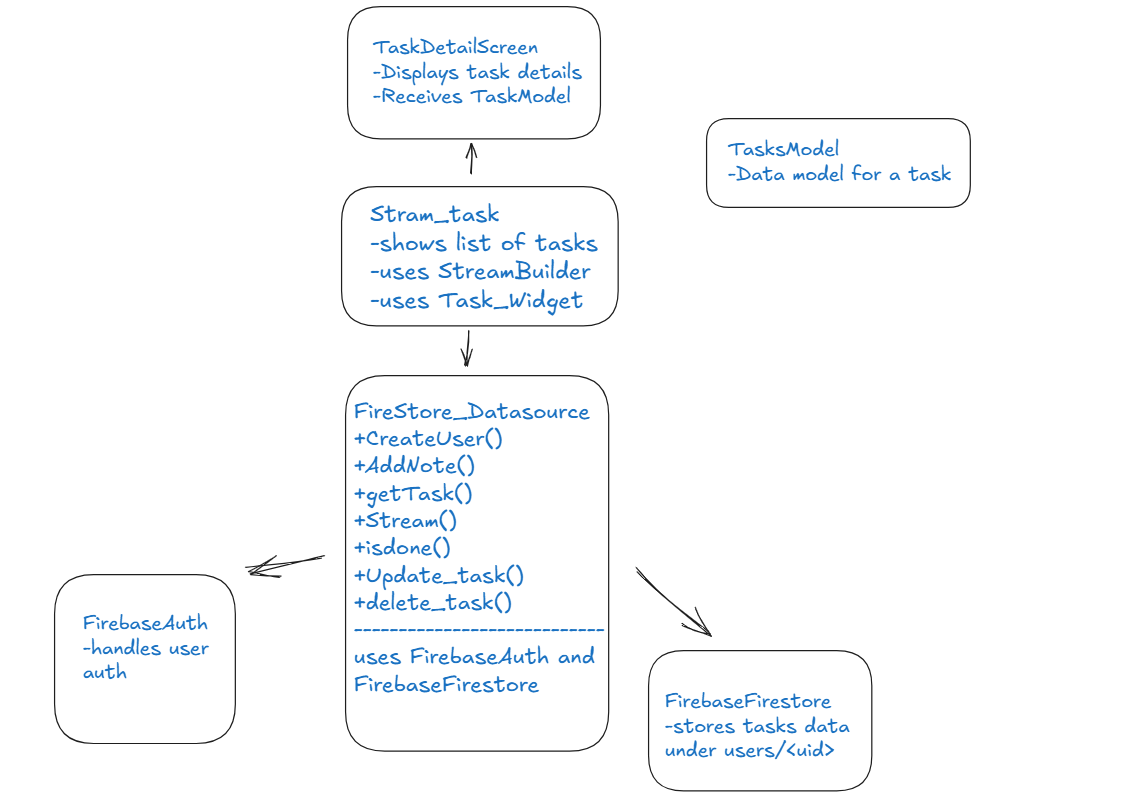
Figure 2: Chatbot Component – Design and Implementation

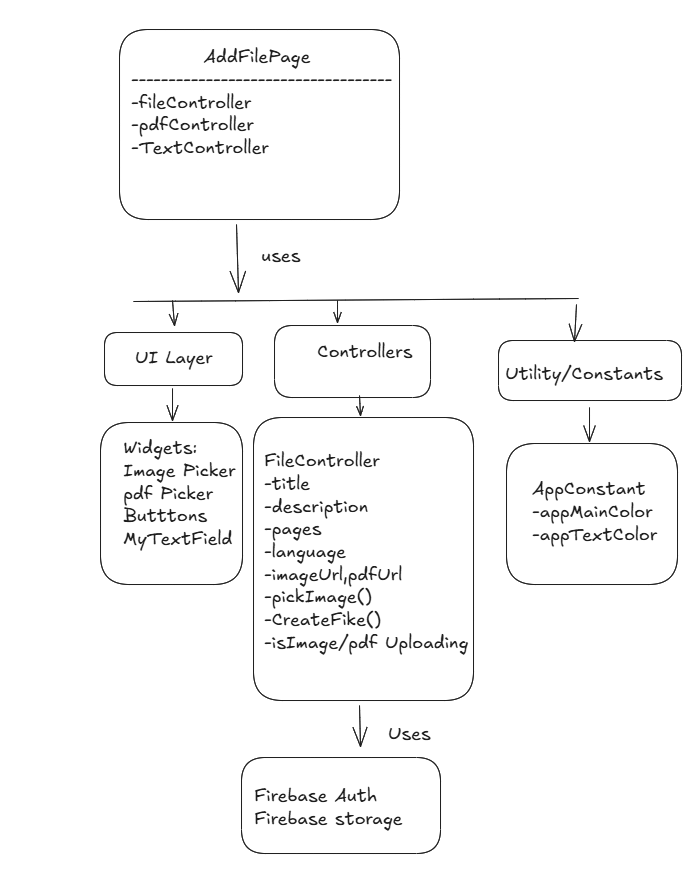
Figure 3: Summarization System

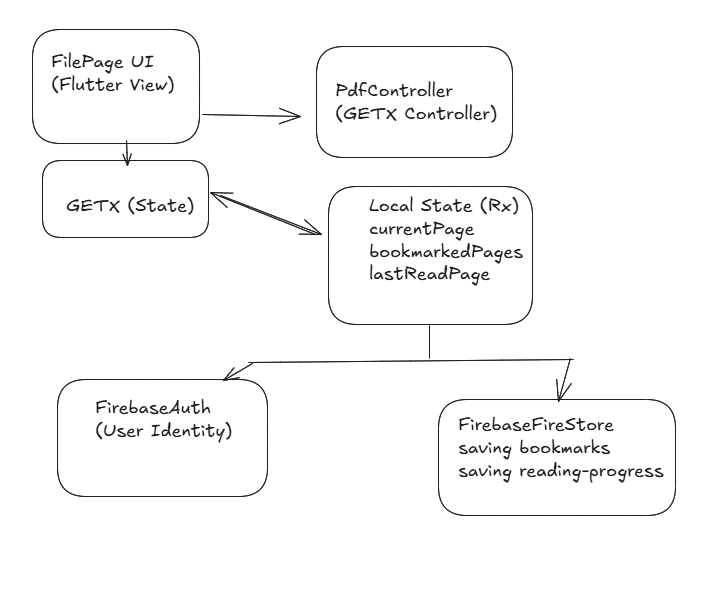


Figure 4 : Translation component

Figure 5:Notes Component

Figure 6 :To Do List

Figure 7 :Add File

Figure 8 : Bookmark & Continue Reading

## 7.Software Implementation

[DO NOT INCLUDE ANY CODE, give details of the software design including the major classes and how the code works]

### 1-Chatbot Component

Major Classes & Their Responsibilities

1.ChatbotScreen , This is the main stateful widget class responsible for the chat UI screen , builds the UI scaffold (AppBar , chat area) , integrates the chat interface widget, holds the list of chat messages , initials services like Gemini.

2.\_ChatebotsscreenState ,This is the state management class , it contains the logic and how messages are created, updated,processed by maintain the list of messages in memory and handle the send and receiving messages, have image picking , editing messages and error handling.

3.ChateMessage , (dash\_chat\_2) , this is a data model class used to represent individual chat messages have fields (text,user,createdAt,medias) and used as chat interface

Also there is ChatUser : this is a simple data model that identifies the sender of a message and contain fields (id, firstName, profileImage) that helps to distinguish between messages from the user and Gemini in the UI .

Last thing Gemini (flutter\_gemini package) this is the AI service class that communicates with

Google’s Gemini API and responsible for processing the user’s input and stream back the AI content.

Now for the Flow of the code :

1.User enter a message and send it

2.A new ChatMessage is created and added to the message list.

3.Gemini process the request and stream back its response.

4.The text is sent to Gemini using the StreamGenirateContent() method

5.As data is received a new ChatMessage is either added or updated with Gemini’s reply.

6.The UI refreshes automatically to reflect the new content .

### 2- Summarization System component

This system organized into layrers:

1.Frontend(Flutter) UI+ Presentation logic

The main class is SummaryScreen is the UI component that displays UI controls (file selector , sliders for summary length/detail and summarize button) also other classes SummartText and PreiewScreen they responsible for displaying the final summary output and if the user want to stores it , and previews the uploaded document before summarization.

2.Service Layer(Flutter)

* SummaryService , this class is the main interface between the UI and backend.It have methods such as request a summary from the backend , store the summary to Firestore for each user,also fetch and delete previous summaries from database.
* .ApiClient , this class responsible for http requests,pares the response with the selected file and user defined parameters and returns the summary to SummaryService class.

3.Backend(Flask)

Flass App (app.py) , defines a single POST endpoint:/api/summarize and on receiving a request it extract the file and parameters from the request ,pass the file to the document processor(document.py) to extract the text,constract a custom prompt using user input(length,detail) then calls the Gemini model to return the summarized outputs as a JSON response.

4.for document processing (document.py) this have process\_document method for detecting the file type and call the suitable method for text extacrion , process\_pdf,process\_word,process\_text these are the method for text extraction to pass it into the summarization pipline.

5.AI model integration (Gemini GenerativeModel) receives prompts and returns a natural language summary as output text

6.Firebase cloud database layer (cloud NoSQL) this stors the summaries per user,each using UID from the FirebaseAuth and for the documentation formate “{filename,summary}

7.This is optional but for the user the application allows to save summaries as pdf into user local storage ,using  permission\_handler package to check whether the required storage permission is granted , This ensures Android’s security and avoids app crashes due to unauthorized access, then using PDF package to create pdf from the summary text , path\_provider package used for saving the generated pdf into external directory ,and after the pdf is generated and saved the application automatically opens it using a compatible app installed on the device all of these actions process throw SnackBar massage to inform user about success pdf downloads or failure , permission denied or any other error

Now for the workflow:

The user select a document and adjust summary setting then click to summarize that will call SummaryService class which delegates to ApiClient class that will sends the file and parameters to Flask , for Flask it will extracts text from file,builds a summarization prompt and sends the request as a query to Gemini model ,then the summary is returned to the app and optionally saved to Firestor.And if the user wants to see the summary it can review it at history or delete it.

### 3.Translation Component

This component have TranslationScreen(StatefullWidget) that holds the dynamic UI behavior and logic and host the layout and UI elements such as language selector,text input field ,translate button and translated text display and \_TranslationScreen contain the core logic and dynamic state that updates UI reactively using setState.

Also GoogleTranslator (from translator package) this is an external service class used to interact with Google translate APIs.

Now for the workflow:

1.Initialization , TranskationScreen class create set of predefined languages default to English

2.The user enter text into TextField , chose source and target languages and the user can make a swap allows flipping between the selected languages.

3.when the user enter “Translate” button the app calls the translation service via GoogleTranslator after it cheaks if the input is valid , then the UI updates to display the result,and the user can copy the text using clipboard .

### 4.Notes Component

This component follows a layered architecture a presentation layer,application logic layer,data layer

The major class is NoteScreen(StatefullWiddget) that opens a dialog for creating and editing notes and resonds to user actions like adding,editing,deleting notes .

Other class is NodeModel acts like data accress layer that interacts directly with Firebase

(Firestore and Authentication) and uses the currents user’s UID (from FirebaseAuth) to access that specific use’s notes.

Now for the workflow :

1.user open NoteScreen,that use NoteMode.getNotesStream() provides a real-time list of notes via Firestore and displayed usong a scrollable ListView.

2.The user can taps Add ,delete ,or Edit then a dialog appears to input note text.Then the note is either added or updated in Firestor because Firestore sends the latest data via stream and UI updates automatically

### 5.To Do List Component

This component also follows a layered architecture with 3 main layers,UI widgets,data layer,model layer.

The major class is TaskDetailScreen(UI layer) that shows task image,title,subtitle,time,statu then takes a Taskmodel object as input without handling any data manipulation.

For data access layer FireStore\_Datasource class acts as bridge between the application and Firebase servies. It create,add,update,delete tasks and provide a real-time stream via Stream<QuerySnapshot> ,for access current user it use FirebaseAuth and for reading and writing tasks under the current user’s document use FirebaseFirestore.

The Model layer TaskModel this responsible for defines the structure of a task the fields are id,title,subtitle,time,image,isDone.

Other class Stream\_task() this class uses a StreamBuilder to listen for task updates, converts Firestore snapshots into a list of TaskModel,shows a message when no tasks are found.

Class TaskDetailScreen, this class displays full details for a selected task.

Now for the work flow :

After the user logs in using FirebaseAuth this will provides the current user ID , the user creats a task using FireStore\_Datasource.AddNote() stores the task under the user’s collection , The Stream\_task class listens via FireStore\_Datasource.stream() to get real-time updates, Snapshot is parsed into TaskModel object with getTask(), user clicks a task and this will navigates to TaskDetailScreen to display all the task details, if the user deletes or updates a task also will be with the help of the FireStore\_Datasource methods.

### 6.Add File Component

This page allows users to upload file cover image, attach a PDF file , input file related metadate (title,discreption,language,total pages)

Major class is AddFilePage (UI layer) this is responsible for building UI layout and connects user actions to controller logic,

FlieController is the state manager that manage text input and upload process ,Contains createFile() method to finalize and submit the file metadate to Firestore.

Now for the work flow :

1.AddFilePage is rendered

2.User click on image placeholer that will call PickImage() method from FileController

3.User click on upload PDF that will call PickPDF() method from FileController

4.User fill all metadata then click on POST button that will call createFile() method from FileController this will typically stored in Firebase Firestore and Storage

### 7. Bookmark & Continue Reading

These features will allow users to save the current page of PDF they are reading (Bookmark) and pick up where they left off after closing and reopening the app (Continure Reading) , also these features will integrated with Firebase Auth and Firestore for authentication and storing data.

In pdfController GETX Controller will manage the PDF reading state(it observes changes to the current page without requiring manual refreshes) , including page navigation , bookmarking and storing the last read page in Firestor.This Controller will use methods such as

addBookmark(pageNumber: int),saveLastPage(pageNumber:int),loadLastPage() from Firestore.

For UI , pdfViewer displays the PDF and allows user interaction.Using pakages like flutter\_pdfview or pdf\_flutter .

Rx variables(states) like currentPage,bookmarkedPages,lastReadpage those holds information for the current page .

Now for the work flow:

1.when the application starts , the pdfController cheaks if a user is logged in by callinf FirebaseAuth.getCurrentUser(),if so the app fetches their UID and proceeds to load the reading progress and bookmarks from Firestor.If not the user will have to sign in before proceeding.

2.The pdf is displayed using Flutter\_pdfview package.

3.when the user taps the bookmark button addBookmark(pageNumber ) method is called , this method saves the page number to Firestore and the state bookmarkPages is updated reactively and the UI reflects this change.

4.loadLastPage() is called when the user open file, to fetch the last read page

## 8. Project Planning and Task Definition

### 8.1 Task Identification

Task Breakdown:

1. Research and Design Phase:
   * Define Requirements: Specify the app's features (file management, summarization, chatbot, etc.).
   * UI/UX Design: Create wireframes of the app according to the user interface and user experience.
   * System Architecture Design: Define the technical architecture (how Python and Flask will interact with Flutter, Firebase usage, etc.).
2. Development Phase:
   * Setup Development Environment: Install and configure all the needed tools (Flutter, Firebase, Python, Flask, etc.).
   * Create Core App Structure: Develop the core structure for navigation, layout, and interaction between Flutter and Python.
   * Backend Development (Flask & Firebase):
     + Code API endpoints for file summarization
     + Implement Firebase Authentication and Firestore.
   * Frontend Development (Flutter):
     + Develop the user interface as per the UI/UX design.
     + Include PDF management through Flutter packages
     + Add "Continue Reading" feature, bookmarks, search, and uploading PDFs.
   * AI Features:
     + Implement Gemini API for chatbot functionality and summarization feature.
     + Test and refine AI response to the chatbot.
   * Localization (Translation): Add translator widget using (translator package: Free Google Translate API for Dart)
3. Testing Phase:
   * Unit Testing: test individual components of both the backend and frontend.
   * Integration Testing: ensure that the app works seamlessly across all features (Firebase, AI, PDF handling).
   * User Testing: conduct testing with a small group of users to gather feedback on the app’s usability.
4. Deployment Phase:
   * Launch Preparation: finish the app for release on relevant platforms.
   * Deploy the App: deploy the app and make sure that Firebase services are enabled.
5. Documentation:
   * Technical Documentation: document out the code and explain the app's features, structure, and APIs.
   * Project Report: begin writing the final report, which should include planning, design, implementation, and results chapters.

### 8.2 Timeline

Week 1-2: Design & Research

* Create app requirements and core functionalities.
* Start UI/UX design (wireframes, flow).
* Define system architecture and tech stack.
* Research Firebase and AI integration.

Week 3-5: Backend & Frontend Setup

* Setup development environment (Flutter, Python, Firebase).
* Design and code basic structure of the app (Flutter navigation, initial screens).
* Set up Firebase Authentication and Firestore.
* Code initial backend APIs in Flask.

Week 6-8: Core Functionality Development

* Code file upload, note-taking, to-do list ,translation, functionalities.
* Begin incorporating AI features (summaries, Gemini API chatbot).
* Add PDF features (searchable PDFs, highlighting, etc.).

Week 9-10: Testing

* Perform unit tests and integration testing.
* Adjust AI chatbot responses.
* User testing for usability and feedback.

Week 11-12: Finalizing & Documentation

* Finish all features, with seamless integration.
* Prepare app for deployment (test on real devices).
* Complete technical documentation.
* Begin writing the final project report.

## 9. Literature Review

### 9.1 Introduction

The review of literature discusses earlier work in four important areas:

1.Ai base content tools =>summarization and chatbots like Gemini API .

2. Document management system such as search , book mark , continue reading , pdf annotation

3.translation and multilingual support through Google translator package integration

4.productivity tools including note taking and making to do list

### 9.2 AI-Based Summarization and chatbots

Summarization software and chatbots are rarely integrated into productivity applications and  
no document management integration so that chatbots cannot leverage it (Adamopoulou & Moussiades, 2020)

The Gaps Filles: our application includes Gemini API for live summarization and chat  
All in one location unlike stand-alone tools.

### 9.3 Document Management and search

Applications such as Adobe acrobat have annotation/bookmark support but no continue reading functionality  
So the Innovation is file uploads and session saving combined

### 9.4 Translation in Productivity tools

Google Translation API is widely used but typically as standalone service Few apps integrate it along with notes taking or takes lists

So Improvement is direct integration for Google translator package allows in app translation without switching platforms.

### 9.5Note Taking and Task Management

These productivity solutions are also not typical to be paired with document management so the improvement is to create them all in one platform + storing them not at the hive but at the firecloud Firestore data base.

### 9.6 conclusion

The review reveals that there is a gap in the market for multifunctional productivity tools that merge AI(summarization and chatbot),document handling, task monitoring. Current tools are fragmented, requiring one to navigate from platform to platform. This project bridges such gaps with a cross functional ai (Gemini), translation, document handling like upload search delete counties read add bookmarks and single workspace like notes and tasks.

## 10. Preliminary Design

### 10.1 Concept

The app is designed to be simple yet powerful digital document management software. It uses AI to help users work more efficiently and faster. For example, it can create a short summary of long PDFs, translate text to other languages, and even respond to questions on documents through an AI chatbot. Users can also take notes, create to-do lists, and search their PDFs easily. To ensure everything is secure and up-to-date, the app is connected with Firebase, which ensures real-time syncing on devices and secure login authentication. In short, the concept is to provide a simple-to-use, one-platform experience that makes document handling simpler and more efficient.

### 10.2 Concept Evaluation and Selection

Different design concepts were carefully analyzed to create the most efficient and user-friendly app. Key considerations of user experience, performance, scalability, security, and accessibility were all taken into account. The goal was to create the interface as smooth and interactive as possible along with instant AI response and PDF loading times. The application also had to deal with large documents efficiently and provide data security, so Firebase Authentication was utilized for secure user management. All features like translation also make the app usable for users who use different languages. After comparing all alternatives, the final implementation uses Google Gemini API for AI parts and Firebase for backend services since this combination offers the best combination of functionality, reliability, and price.

### 10.3 Design Constraints

The initial app design had to run under certain constraints. One, real-time execution was essential AI features like summarization and chatbot response needed to be rapid with minimal delay. Two, network reliance in that capabilities like AI summarization, chatbot, open e-file require internet connectivity to work. Finally, the security of the data was top priority, and for that reason the app utilizes Firebase Authentication and strict Firestore database rules to secure the privacy of users. These restrictions impacted the creation of the app to render it quick, effective, and safe.

### 10.5 Preliminary Analysis

Before release, we extensively tested the app to make sure it performs fine and performs user functions. AI summarization accuracy and speed was tested against Google Gemini API standards. Firebase was load-tested to ensure it handles heavy traffic without synchronization issues. The user interface was fine-tuned to be seamless on different devices, from low-end phones to high-end smartphones. We also developed robust error handling with extensive logging to be able to identify and resolve any issues immediately. From these tests, the app is well on its way to providing a speedy, consistent experience with correct AI support that users will find easy to use and useful.

## 11. Detailed System Design

### 11.1 Engineering Analysis and Simulation

The App's complete system design was exhaustively analyzed for performance, reliability, and user satisfaction. Several simulations and performance tests were conducted, including:

* AI Model Accuracy: Validation of the summarization and question-answering ca[ability of the Google Gemini API on multi-varying datasets.

• System Load Simulation: Stressed Firebase’s backend to observe response time of database under heavy load of users

• User Interface Performance:  Tested rendering performance of Flutter in order to present smooth interactions and minimize latency.

• Error Recovery and Logging: Strong error-handling coupled with bulk-scale log generation for debugging.

### 11.2 Layouts, Drawings, Equipment Specifications

**System Architecture Diagram: figure 9.2**

****

**Database schemas:**

Firebase (NoSQL – Cloud Firestore / Realtime Database)

Data Type: Semi-Structured / Unstructured (JSON Format)

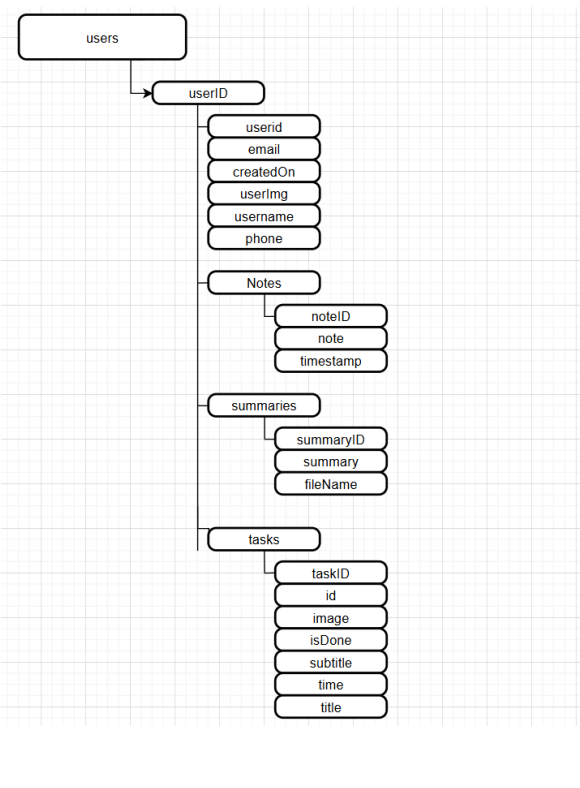
* Firebase stores data in documents (Firestore) or JSON trees (Realtime Database).

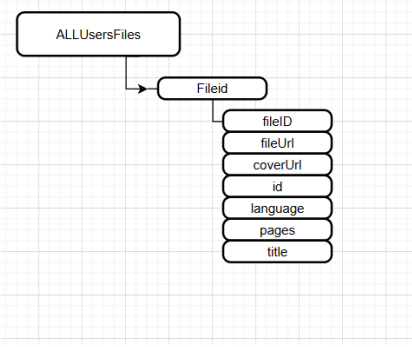
• Data is stored in key-value pairs, with flexible structure.

• Firebase supports: Flexible schema don’t need to define the structure before inserting data

A diagram of a computer

AI-generated content may be incorrect.Figure 2 :userFile

Figure 1:Users

Figure 3 :AllUsersFiles

**Component Overview:**

* **Frontend:** Built with Flutter, with widgets for document viewing, note-taking, task creation, translation, document uploading,
* **Backend:** Python-built using Flask for API request and response management.
* **Database:** Firebase Firestore to store and synchronize data in real-time.
* **AI Integration:** Google Gemini API for summarization and chatbot services.

**Layout Design:**

* **Home Screen:** Displaying uploaded files
* **Drawer** for accessing all the tools(add file, summary,translation,notes,todolist)

### 11.3 Applicable Codes and Standards Used in the Design

The app adheres to the following standards and best practices:

* **ISO/IEC 25010:** Ensuring software quality and maintainability.
* **IEEE 830:** Standard for software requirements specifications.
* **ISO 27001:** Implementing information security best practices.
* **REST API Standards:** Ensuring seamless communication between the backend and external services.

### 11.4 Economic Analysis

Economic viability of the app is approximated by estimating development, deployment, and maintenance expenses. Development expenses include Flutter (frontend) and Python Flask (backend) development expenses, integration expenses of Google Gemini API, and Firebase services (auth and Firestore database). Cloud and hosting costs are approximated as a function of Firebase Firestore storage needs and API call frequency, which increase with the number of users. Maintenance expenses post-launch involve regular updates, bug fixing, and potential feature enhancements.

The project is economically viable as long as the costs are maintained and minimized through Firebase's pay-as-you-go pricing model and affordable scalability. Any subsequent development will be dictated by users' feedback and market needs for the purpose of balancing profitability and functionality.

## 12. Implementation

### 12.1 Construction

The construction stage of the App involved assembling the various parts and ensuring there is seamless integration. The key parts constructed are:

1. **Frontend:** Constructed using Flutter for cross-platform compatibility on Android. PDF readers, AI chat screens, note-taking, task lists, translation interfaces were developed using customizable widgets.
2. **Backend:** Constructed using Python and Flask to handle API requests, data processing, and external AI service calls.
3. **Database:** Firebase Firestore was used for real-time data storage and synchronization.
4. **AI Integration:** Google Gemini API was integrated for summarization and chatbot functionality.

The build process consisted of an **agile methodology**, with iterative development and testing.

### 12.2 Programming

Development consisted of some necessary steps to conceptualize a usable and functional app. Firstly, the frontend was built considering Flutter, keeping in mind minimal and responsive designs. State management techniques were adopted by the developers to ensure sleek functionality and envisioned modular components that would make upgrades and maintenance more convenient in the future. In the backend, API endpoints were established using Flask to enable front-end and backend system interaction. Proper error handling and logging were implemented to improve reliability and ease debugging. The AI features were integrated through the Google Gemini API, which provided suitable text summarization and chatbot functionality. API requests and responses were handled with attention by the development team in order to remain efficient. Firebase was used for storing data in real-time using Firestore, while Firebase Authentication defended user access securely.

Finally, thorough testing was conducted, including unit tests on each component and integration tests to verify that the frontend, backend, and AI services all collaborated perfectly together. This step was crucial in delivering a high-quality and stable application.

### 12.3 Validation

Validation process ensured that the application met both functional and non-functional needs. Several key tests were conducted to ensure performance, security, and usability.

**1.Functional Testing** was aimed at major features like PDF uploads, summarization via AI, chatbot response via AI, and note-taking functionality. Test documents were used to ensure accuracy and reliability of the summaries produced by AI.

**2.Performance Testing** validated the app's response on different devices and networks.

**3.Security Testing** ensured that Firebase Authentication worked as expected, protecting users' data and limiting unauthorized access.

Finally, User Acceptance Testing (UAT) involved real users in testing the app for feedback. Their input acted to raise usability concerns, which led to iterative improvements before the final release.

The results of the tests confirmed that the application performed as intended, meeting all design requirements and having an impeccable user experience.

## 13. Results and Discussion

### 13.1 Summary of Goals Met by the Design

The application successfully fulfilled its main objectives, delivering a stable and user-friendly platform for digital document management. Through the integration of the Google Gemini API, the application delivers accurate and concise AI-based summarization, significantly enhancing productivity. The integrated chatbot further improves usability with direct, relevant responses to user queries, making document interaction more natural and effective. Document management capabilities were at the core, with seamless PDF uploading, viewing, and searching. The app supports searchable PDFs, making it easy for users to search and highlight important terms within their documents. This feature streamlines information access, saving users time and effort. For enhanced productivity, the application features integrated note-taking and to-do list capabilities, all synced in real-time via Firebase. The "Continue Reading" and bookmarking feature enhance usability by enabling effortless navigation and progress tracking across documents.

Data security and integrity were made a top preference leveraging Firebase Authentication, which safeguards user accounts, and Cloud Firestore, which enforces consistent real-time data integrity. Collectively, they create a secure and stable environment to handle confidential documents.

Overall, the app combines cutting-edge AI functionality with practical productivity tools in a cross-platform application. Future releases will focus on expanding AI functionality and further optimizing performance to meet the evolving demands of users. Effective incorporation of such functionality demonstrates the potential of the app as a comprehensive document management solution.

### 13.2 Summary of Constraints and Codes Met by the Design

The design adhered to multiple constraints and standards to ensure quality, security, and usability.

**Constraints Met:**

* **Performance:** Capture low-latency AI responses and smooth PDF rendering.
* **Security:** Firebase Authentication guarantees user data security.
* **Storage Management:** Utilized efficient storage and management of user data using Firebase Firestore.
* **Network Dependency:** API requests were optimized to decrease network load and improve response time.

**Codes and Standards Met:**

* **ISO/IEC 25010:** Ensured software quality, usability, and maintainability.
* **IEEE 830:** Followed requirements documentation standards.
* **ISO 27001:** Applied security best practices to protect user data.
* **REST API Standards:** Ensured consistent communication between the backend and APIs.

## 14.Conclusion

Here in this project, we aimed to create and design an AI-powered interactive digital content application that enhances the e-reading experience through smart features. With the implementation of AI-powered summarization, user support chatbot, and other productivity tools, the app strives to provide an uninterrupted reading and management experience.

During the course of development, we encountered and resolved numerous issues, including integrating APIs, syncing data in real-time, and designing an user-friendly interface. With technologies like Flutter for client-side, Python with Flask for server-side, and Firebase to handle data, we were successful in creating an efficient and scalable app.

The AI functionality, particularly the summarization and chatbot capabilities via the Gemini API, added considerable value in enabling quick content comprehension and the procurement of related information on behalf of the users. The incorporation of a translation widget, note-taking capabilities, and to-do lists also helped towards improving the application's usability.

From user review and testing, the app demonstrated effectiveness in simplifying content consumption complexity and enhancing productivity. There is always room for improvement, though. Future enhancements can involve extended language support for translation, optimizing chatbot output, and offline support.

In short, the project was successful in what it set out to achieve through offering a working and user-friendly digital content management platform. It shows the capability of AI to enhance digital experience and paves the way for further innovations in e-reading.

## 15. Recommendations

Recommendations for Future Improvements and Expansions based on the knowledge and experience gained during the development and piloting of the AI-Based Interactive Digital Content Application, the following are the recommendations for future improvements and expansions:

1. **Enhance AI Capabilities**: Enhance the accuracy and contextual awareness of the AI-powered chatbot and summarization features. Adding more advanced natural language processing (NLP) models can enhance the user experience.
2. **Expand Language Support**: Extend the translation widget to support additional languages, making the app usable by a global user community.
3. **Implement Offline Mode**: Offer offline functionality for certain features, such as viewing previously summarized content, saved notes, and bookmarked PDFs. This will render it more functional where connectivity is poor.
4. **Advanced Search and Filter Options**: Implement additional advanced search functionality, including keyword suggestions and voice search, to facilitate easy content discovery.
5. **User Customization**: Provide options for users to customize the look and feel of the app, font size, reading themes, and layout options for a personalized reading experience.
6. **Multi-Platform Support**: Develop desktop and web-based application versions, with a consistent user experience across devices.
7. **Security Improvements**: Incorporate additional security measures, like two-factor authentication (2FA) and more robust data encryption, to protect user information.
8. **Integration of User Feedback**: Design a feedback system within the app, where users can report issues and suggest features for subsequent updates.
9. **Video summarization** : Make the users not only summarize files but also videos.

By implementing these recommendations, the application will become more powerful, interactive, and user-centered, resulting in wider adoption and satisfaction among its users.

## List of References

1. Google AI. (2024). *Gemini API Documentation*. Retrieved from <https://ai.google.dev/gemini-api/docs>
2. Firebase. (2024). *Firebase Documentation*. Retrieved from <https://firebase.google.com/docs>
3. Flutter. (2024). *Flutter Documentation*. Retrieved from <https://flutter.dev/docs>
4. Python Software Foundation. (2024). *Python Documentation*. Retrieved from <https://docs.python.org/3/>
5. Flask. (2024). *Flask Documentation*. Retrieved from <https://flask.palletsprojects.com/>
6. Stack Overflow. (2024). *Developer Community Discussions*. Retrieved from [https://stackoverflow.com](https://stackoverflow.com/)

## Appendices

**Appendix A: Project Timeline**

* [Detailed Gantt chart showing the project schedule, milestones, and task completion.](https://docs.google.com/spreadsheets/d/1SU3fzE3tLUfa1zmbPBf7gqU1kRLA3boO/edit?usp=sharing&ouid=104868842639778942632&rtpof=true&sd=true)

**Appendix B: System Requirements**

**1. Development Environment**

**-Hardware Requirements:**

- Processor: Intel Core i5 or equivalent (minimum)

- RAM:8GB (minimum) / 16GB (recommended for smoother performance)

- Storage:10GB of free space (for IDEs, SDKs, and dependencies)

- Operating System: Windows 10 pro/11,

- **Software Requirements**:

- Flutter SDK (Flutter --v 3.24.5)

- Dart SDK (bundled with Flutter)

- Android Studio (for Android emulation) (Android SDK version 35.0.1)

- Firebase CLI & Tools(for authentication and Firestore integration)

- Python 3.13.0 & Flask (for backend API development)

- Google Gemini API Key (for AI-powered features)

**2. Testing Environment**

- Physical Devices:

- Android: Devices running Android 11+ (e.g., Samsung Galaxy , Google Pixel 7a)

- Network Conditions:

- Wi-Fi 4G /4G+/5G

- Cloud Services:

- Firebase Project (for authentication, Firestore, and hosting)

- Google Cloud Platform (GCP) (for Gemini API access)

**3. Deployment Requirements**

- Mobile Platforms:

- Google Play Store (for Android APK)

- Apple App Store (for iOS IPA)

- Backend Hosting:

- Cloud Services: Firebase Hosting / Render (for Flask API)

## Meeting Minutes

From the first meeting through which the team decided to work with Flutter, Python, and Firebase while envisioning early AI research, there was a standard development process through which the project progressed. The gathering of requirements, the conversations about UI needs, and features like chatbot and summarization were the focus of follow-up meetings. Regular reviews of frontend and backend issues focused on enhancing UI and working with APIs. User feedback captured from test sessions improved AI response quality and usability. The team thoroughly reviewed and created documentation and presentation materials for submission, and finalized revisions with respect to supervisor feedback. Clear action items were included in each stage to guarantee consistent progress toward project handover.