**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**](#_1._Acknowledgment)**…………………………..………………………………………...**[**5**](#page5) **1.**

**2.** [**Abstract**](#_2._Abstract)**……………………………………………………….…………...................**[**5**](#page5)

**3.** [**Introduction**](#_5._Introduction)**…………………………………………………………. ……………...**[**6**](#page5)

3.1 [Statement of the Problem](#_3.1_Statement_of) …………………………………………………. ….....[6](#page5)

3.2 [Significance of the Project](#_3.2_Significance_of) ……………………………………………………….6

3.3 [Goals](#_3.3_Goals_) ……………………………………………………………………………...6

3.4 [Contemporary Issues](#_3.4_Contemporary_Issues) ……………………………………………………………...7

3.5 [Impact on Society](#_3.5_Impact_on) …………………………………………………………...……7

3.6 [Initial Constraints](#_3.6_Initial_Constraints) ……………………………………………………………....…8

**4.** [**Professional Practice Constraints**](#_4._Professional_Practice)**……………......…………………. .......................**[**8**](#page5)

4.1 [Manufacturability Constraints](#_4.1._Manufacturability_Constraints) ………………………………………...………….8

4.2 [Economic Constraints](#_4.2._Economic_Constraints) …………………………………………………………….9

4.3 [Sustainability Constraints](#_4.3._Sustainability_Constraints) …………………………………………………………9

4.4 [Environmental Constraints](#_4.4._Environmental_Constraints) ………………………………………………………..9

4.5 [Health and Safety Constraints](#_4.5._Health_and) …………………………………………………….9

4.6 [Ethical Standards Constraints](#_4.6._Ethical_Standards) …………………………………………………….9

4.7 [Social Values Constraints](#_4.7._Social_Values) …………………………………………………………9

**5.** [**System Architecture and Design**](#_5._System_Architecture)**……………………………………. ......................**[**10**](#page5)

**6.** [**Software Implementation**](#_6._Software_Implementation)**………………………………...……………....................**[**14**](#page5)

6.1 [Chatbot Component](#_4.1._Manufacturability_Constraints) ………………………………………………………………14

6.2 [Summarization System Component](#_6.2_Summarization_System) ……………………………………………...14

6.3 [Translation Component](#_6.3_Translation_Component) …………………………………………………………..16

6.4 [Notes Component](#_6.4_Notes_Component) …………………………………………………………………16

6.5 [To Do List Component](#_6.5_To_Do) …………………………………………………………….16

6.6 [Add File Component](#_6.6_Add_File) ………………………………………………………………17

6.7 [Bookmark & Continue Reading](#_6.7_Bookmark_&) …………………………………………………..17

**7.** [**Project Planning and Task Definition**](#_7._Project_Planning) **………………………………. ……………..[18](#page5)**

7.1 [Task Identification](#_7.1_Task_Identification) …………………………………………………………………18

7.2 [Timeline](#_7.2_Timeline) ……………………………………………………………………………20

**8.** [**Literature Review**](#_8._Literature_Review)**……………………………………………………. ………………...**[**21**](#page5)

8.1 [Introduction](#_8.1_Introduction) ………………………………………………………………………….21

8.2 [AI-Based Summarization and Chatbots](#_8.2_AI-Based_Summarization) ……………………………………………...21

8.3 [Document Management and Search](#_8.3_Document_Management) ………………………………………………….21

8.4 [Translation of Productivity Tools](#_8.4_Translation_of) …………………………………………………….21

8.5 [Note Taking and Task Management](#_8.5_Note_Taking) ………………………………………………….21

8.6 [Conclusion](#_8.6_conclusion) ……………………………………………………………………………21

**9.**  [**Preliminary Design**](#_9._Preliminary_Design)**…………………………………………………. …………………**[**22**](#page5)

9.1 [Concept](#_9.1_Concept) ………………………………………………………………………………22

9.2 [Concept Evaluation and Selection](#_9.2_Concept_Evaluation) …………………………………………………...22

9.3 [Design Constraints](#_9.3_Design_Constraints) …………………………………………………………………..22

9.4 [Preliminary Analysis](#_9.4_Preliminary_Analysis) …………………………………………………………………22

**10.** [**Detailed System Design**](#_10._Detailed_System) **………………………………………………. ……………….**[**23**](#page5)

10.1 [Engineering Analysis and Simulation](#_10.1_Engineering_Analysis) ………………………………………………23

10.2 [Layouts, Drawings, Equipment Specifications](#_10.2_Layouts,_Drawings,) ……………………………………..23

10.3 [Economic Analysis](#_10.3_Economic_Analysis) …………………………………………………………………25

**11.** [**Implementation**](#_11._Implementation) **…………………………………………………………. …………….**[**25**](#page5)

11.1 [Construction](#_11.1_Construction) ………………………………………………………………………...25

11.2 [Programming](#_11.2_Programming) ……………………………………………………………………….26

11.3 [Validation](#_11.3_Validation) …………………………………………………………………………...26

**12.** [**Results and Discussion**](#_12._Results_and) **…………………………………………………………. ……..[26](#page5)**

12.1 [Summary of Goals Met by the Design](#_12.1_Summary_of) ……………………………………………...26

**13.** [**Conclusion**](#_13._Conclusion) **…………………………………………………………. …………………..[27](#page5)**

**14.** [**Recommendations**](#_14._Recommendations) **………………………………………………………. …………….**[**28**](#page5)

[**List of References**](#_List_of_References) **…………………………………………………………. ………………**[**29**](#page5)

[**Appendices**](#_Appendices) **…………………………………………………………. ……………………...**[**29**](#page5)

[**Meeting Minutes**](#_Meeting_Minutes_1) **…………………………………………………………. ………………..[30](#page5)**

[**User Manual**](#_User_Manual) **…………………………………………………………. ……**……………….[**30**](#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

## **1. 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 also like to extend a special appreciation to the Jordan University of Science and Technology (JUST) and the Computer Engineering Department (CPE) for offering us the means and information necessary to make this project a reality. 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 everyone who directly or indirectly helped make our project successful. Your guidance and support have been most essential in turning our dream into a reality.

## **2. 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 a 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.

## **3. Introduction**

### **3.1 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 altogether.

### **3.2 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, note taking, 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.

### **3.3 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. The **translation** screen translates text instantly to other languages.

5. Cloud Sync **saves** files, bookmarks, last position to continue reading, notes, and to-do lists in Firebase, and views 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.

### **3.4 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.

**• 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.

**• Obtaining Information in a Timely Fashion**

Most document programs do not have good search capabilities, and the user wastes a lot of 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 exactly where it is located in the document.

**• 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.

**• 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.

### **3.5 Impact on Society**

**• 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.

**• Ideal for Everyone**

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

**• Keeps You Organized**

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

**• 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.

**• Makes Information More Understandable**

### **3.6 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.

## **4. 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:

### **4.1. 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 above all else, with immediate access to functionality including note-taking, to-do lists, uploading files, and AI features.

### **4.2. Economic Constraints**

The project will be economically viable for developers and users alike:

· Development Costs: Firebase offers a free tire, so initial development with minimal expenditure.· Open-source tools (e.g., Flutter, Flask) are used to minimize expenses.

### **4.3. Sustainability Constraints**

### The process uses sustainable practices: · Firebase's cloud-based system accommodates long-term data storage without the use of physical resources. · The app is minimalistic, saving energy when in operation and being compatible with a wide range of devices (Android).

### **4.4. Environmental Constraints**

Decrease in use of paper by providing electronic means of summarizing files, note-taking, and task management, the app promotes a shift from paper processes to digital ones.  
helping in environmental preservation.

### **4.5. Health and Safety Constraints**

User Data Protection:   
· Firebase Authentication securely logs in the app, keeping the user's details protected from  
unauthorized access.  
· Firestore rules are utilized to protect sensitive information, such as uploaded documents and personal notes.

### **4.6. 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 avoiding any features or functionalities that could be used for malicious purposes.

### **4.7. Social Values Constraints**

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

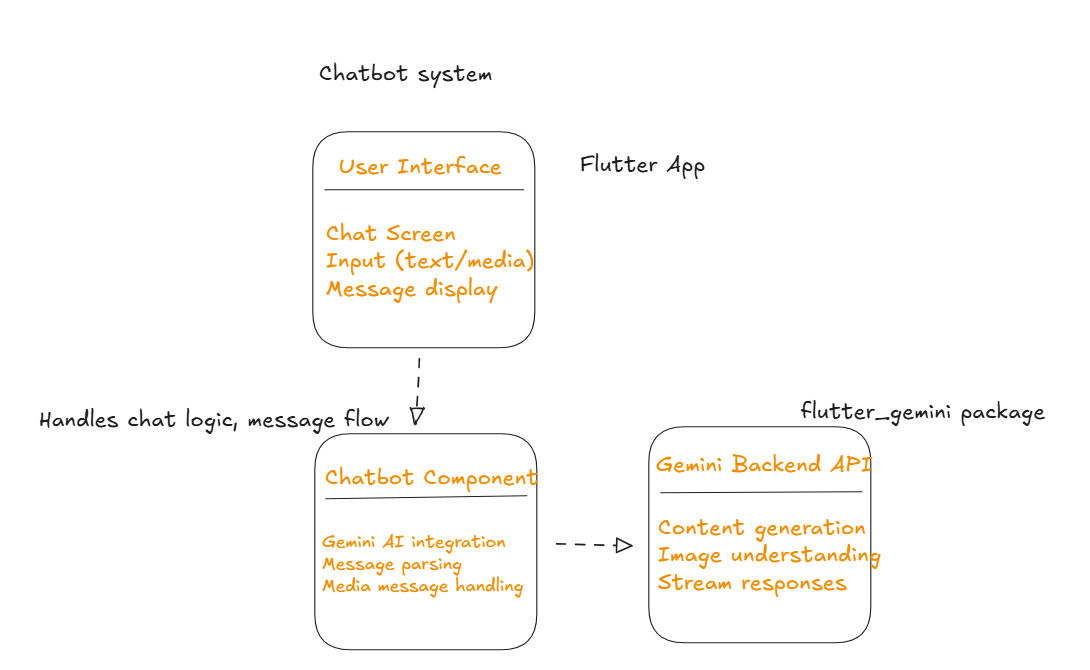
expectations:

**· Ethical Usage**: The app avoids any features that could be exploited for unethical purposes, such as special information spread.

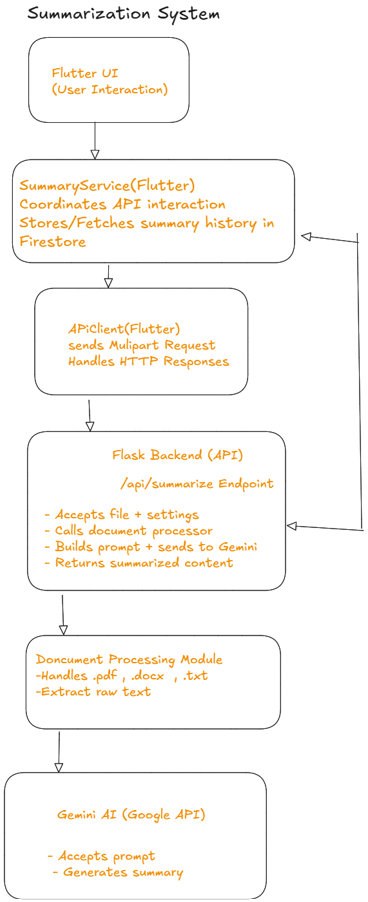
**· Inclusivity and Accessibility**: Designed to cater to a diverse user base, including individual casual readers and academic users.

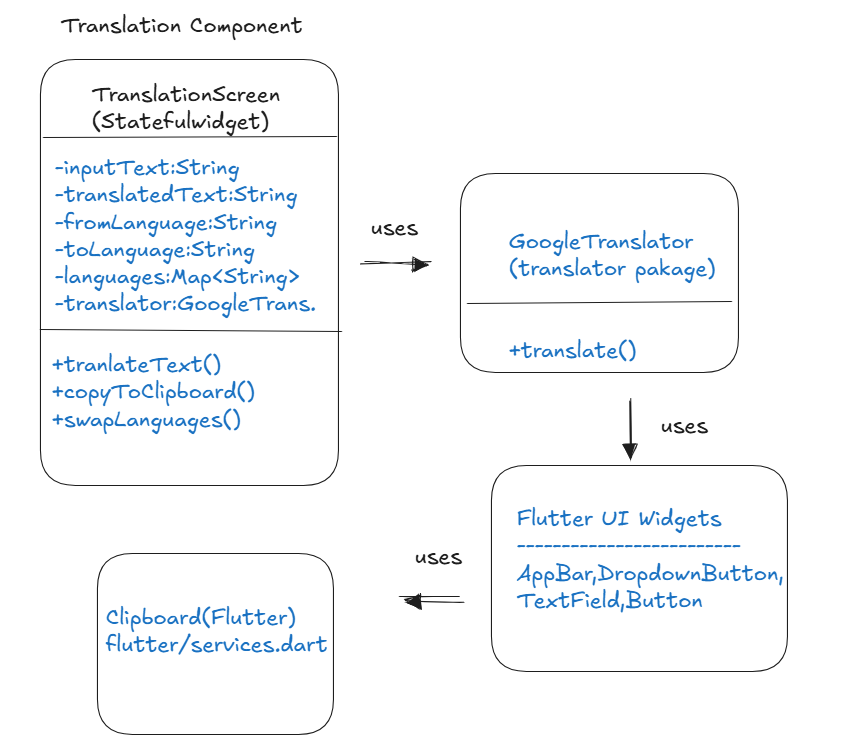
## **5. System Architecture and Design**

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

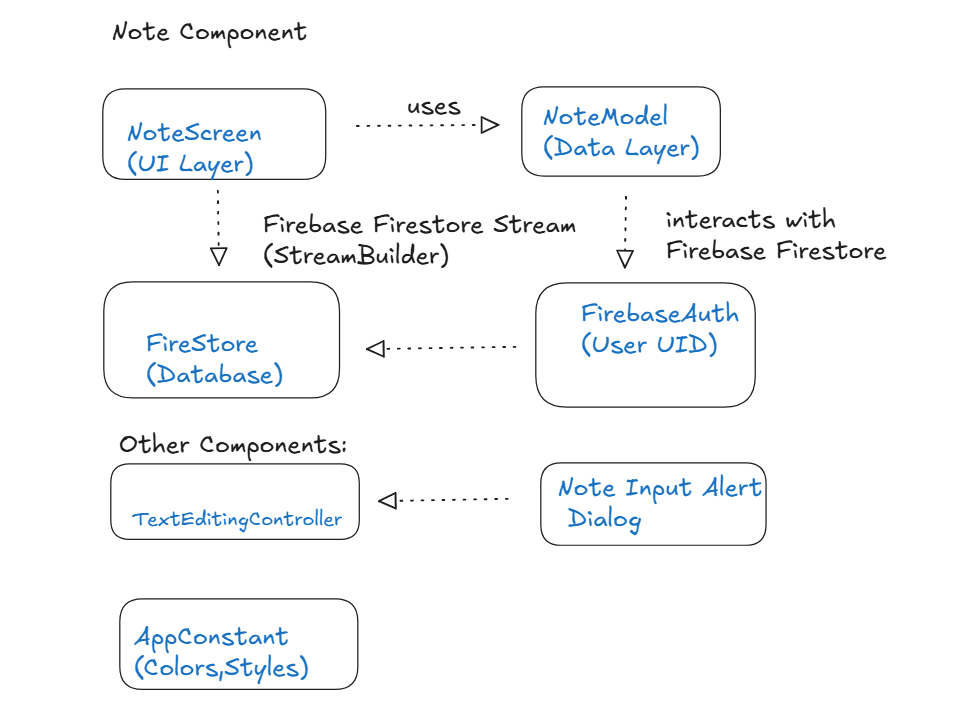
 **Figure 2:** Chatbot Component design and implementation

**Figure 3**: Summarization System

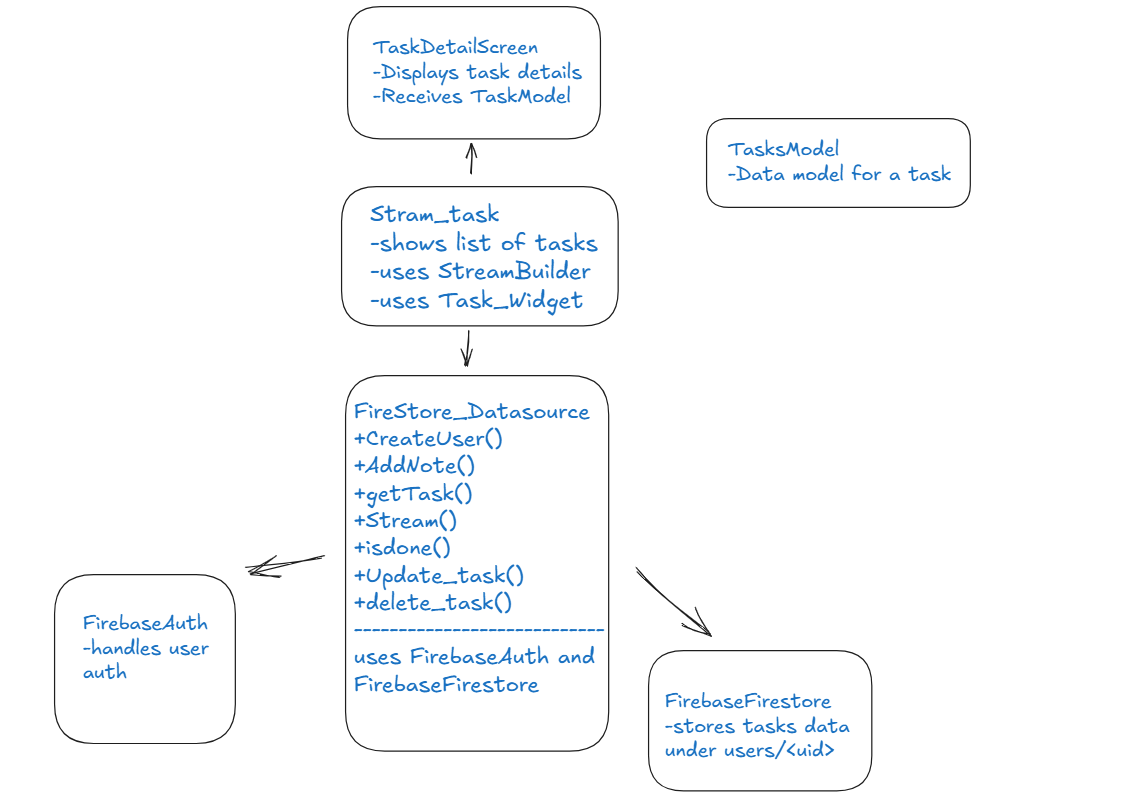




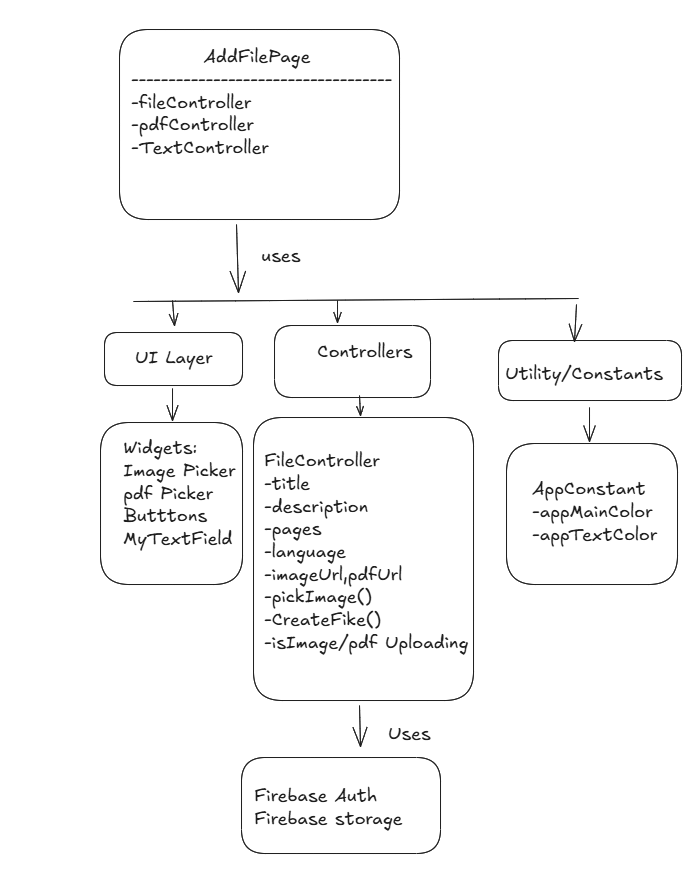
**Figure 4:** Translation component



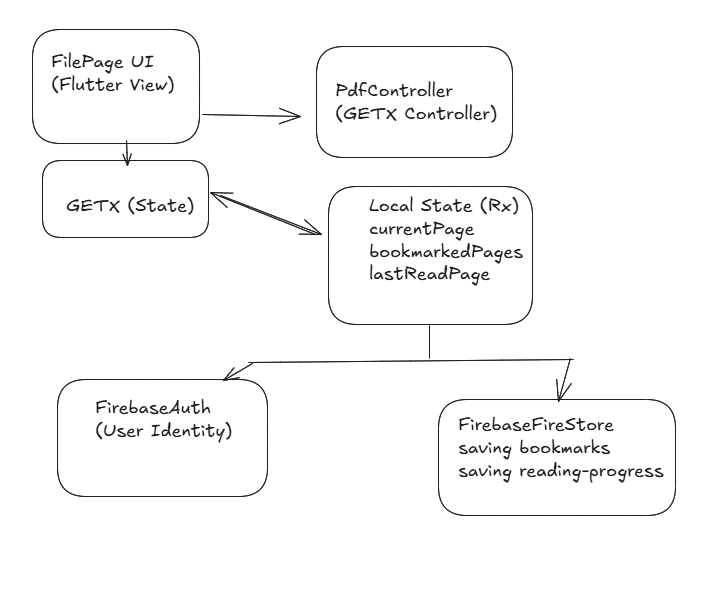
**Figure 5**: Notes Component



**Figure 6**: To Do List



**Figure 7**: Add File

**Figure 8:** Bookmark & Continue Reading

## **6. Software Implementation**

### **6.1 Chatbot Component**

Major Classes & Their Responsibilities

**Chatbot Screen**: 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, and initializes services like Gemini.

**\_ChatebotsscreenState**: this is the state management class, it contains the logic and how messages are created, updated, processed by maintaining the list of messages in memory and handling the sending and receiving messages, have image picking, editing messages, and error handling.

**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 a chat interface

**ChatUser:** This is a simple data model that identifies the sender of a message and contains fields (id, firstName, profileImage) that help to distinguish between messages from the user and Gemini in the UI.

**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 streaming back the AI content.

**Now for the Flow of the code:**

The user enters a message and sends it. A new ChatMessage is created and added to the message list, then Gemini processes the request and streams back its response. The text is sent to Gemini using the StreamGenirateContent() method and as data is received, a new ChatMessage is either added or updated with Gemini’s reply. The UI refreshes automatically to reflect the new content.

### **6.2 Summarization System component**

This system is organized into layers:

**6.2.1. 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 store it , and previews the uploaded document before summarization.

**6.2.2 Service Layer (Flutter)**

• **SummaryService:** This class is the main interface between the UI and the backend. It has methods such as requesting a summary from the backend, storing the summary in Firestore for each user, and also fetching and deleting previous summaries from the database.

• .**ApiClient:** This class is responsible for HTTP requests, parses the response with the selected file and user-defined parameters, and returns the summary to the SummaryService class.

**6.2.3. Backend (Flask)**

• **Flask App (app.py),** defines a single POST endpoint:/api/summarize and on receiving a request it extracts 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.

• 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.

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

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

• 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, they can review it in history or delete it.

### **6.3 Translation Component**

This component has 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 the translator package) is an external service class used to interact with Google Translate APIs.

**Now for the workflow:**

1. Initialization, TranslationScreen class creates a set of predefined languages, defaulting to English

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

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

### **6.4 Notes Component**

This component follows a layered architecture: a presentation layer, an application logic layer, and a data layer. The major class is NoteScreen (StatefulWidget) that opens a dialog for creating and editing notes and responds to user actions like adding, editing, and deleting notes. The other class is NodeModel acts like a data access layer that interacts directly with Firebase (Firestore and Authentication) and uses the current user’s UID (from FirebaseAuth) to access that specific user’s notes.

**Now for the workflow:**

The user opens NoteScreen, which uses NoteMode.getNotesStream() provides a real-time list of notes via Firestore and is displayed using a scrollable ListView. then, the user can tap Add, delete, or Edit, then a dialog appears to input note text. Then the note is either added or updated in Firestore because Firestore sends the latest data via stream, and UI updates automatically.

### **6.5 To Do List Component**

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

The major class is TaskDetailScreen(UI layer) that shows task image, title, subtitle, time, status, then takes a Taskmodel object as input without handling any data manipulation. For the data access layer FireStore\_Datasource class acts as a bridge between the application and the Firebase services. It creates, adds, updates, and deletes tasks and provides a real-time stream via Stream<QuerySnapshot>. For accessing the current user, it uses FirebaseAuth, and for reading and writing tasks under the current user’s document, use FirebaseFirestore.

The Model layer TaskModel is responsible for defining the structure of a task. The fields are id, title, subtitle, time, image, and 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 workflow:**

After the user logs in using FirebaseAuth, this will provide the current user ID, the user creates 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.

### **6.6 Add File Component**

This page allows users to upload a file cover image, attach a PDF file, and input file-related metadata (title, description, language, total pages).

The major class is AddFilePage (UI layer), which is responsible for building the UI layout and connecting user actions to controller logic. FlieController is the state manager that manages text input and the upload process. Contains a createFile() method to finalize and submit the file metadata to Firestore.

**Now for the workflow:**

### AddFilePage is rendered. The user clicks on the image placeholder, which calls the PickImage() method from FileController. then, the User clicks on upload PDF, which calls the PickPDF() method from FileController. finally, the User fills all metadata and clicks on the POST button, which will call the createFile() method from FileController. This will typically be stored in Firebase Firestore and Storage.

### **6.7 Bookmark & Continue Reading**

These features will allow users to save the current page of the PDF they are reading (Bookmark) and pick up where they left off after closing and reopening the app (Continue Reading). Also, these features will be 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 packages like flutter\_pdfview or pdf\_flutter. Rx variables(states) like currentPage, bookmarkedPages, and lastReadPage hold information for the current page.

**Now for the workflow:**

1. When the application starts, the pdfController checks if a user is logged in by calling 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 the 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 opens a file, to fetch the last read page.

## **7. Project Planning and Task Definition**

### **7.1 Task Identification**

Task Breakdown:

**1. Research and Design Phase:**

• Define Requirements: Specify the app's features (file management, summarization, chatbot…).

• 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).

**2. Development Phase:**

• Setup Development Environment: Install and configure all the needed tools (Flutter, Firebase, Python,

Flask).

• 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" features, bookmarks, search, and uploading PDFs.

• AI Features:

▪ Implement Gemini API for chatbot functionality and summarization features.

▪ Test and refine the AI response to the chatbot.

• 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 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.

### **7.2 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 the 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.

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.

## **8. Literature Review**

### **8.1 Introduction**

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

1. AI-based content tools =>summarization and chatbots like Gemini API.

2. Document management systems such as search, bookmark, continue reading, and PDF annotation.

3. Translation and multilingual support through Google Translate package integration.

4. Productivity tools, including note taking and making to do list.

### **8.2 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 Files: our application includes the Gemini API for live summarization and chat  
All in one location, unlike stand-alone tools.

### **8.3 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.

### **8.4 Translation of Productivity Tools**

Google Translate API is widely used, but typically as a standalone service. Few apps integrate it along with note-taking or taking lists. So, Improvement is the direct integration of the Google translator package allows in-app translation without switching platforms.

### **8.5 Note Taking and Task Management**

These productivity solutions are also not typically paired with document management, so the improvement is to create them all in one platform + storing them not at the hive but at the Firestore database.

### **8.6 conclusion**

The review reveals that there is a gap in the market for multifunctional productivity tools that merge AI(summarization and chatbot), document handling, and 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, continue reading, add bookmarks, and a single workspace like notes and tasks.

## **9. Preliminary Design**

### **9.1 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, docx, txt), 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.

### **9.2 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.

### **9.3 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 those capabilities like AI summarization, chatbot, and open e-file requires 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.

### **9.4 Preliminary Analysis**

Before release, we tested the app to make sure it performs fine and performs user functions. AI summarization accuracy and speed were tested. Firebase was load-tested to ensure it handles traffic without synchronization issues. The user interface was fine-tuned to be seamless on different devices. We also developed robust error handling with wide 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.

## **10. Detailed System Design**

### **10.1 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 capability of Google

Gemini API on multi-varying datasets.

•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.

### **10.2 Layouts, Drawings, Equipment Specifications**

**System Architecture Diagram: Figure 9.2**

****

**Database schemas:**

Firebase (NoSQL – Cloud Firestore / Realtime Database)

Data Type: Unstructured (JSON Format)

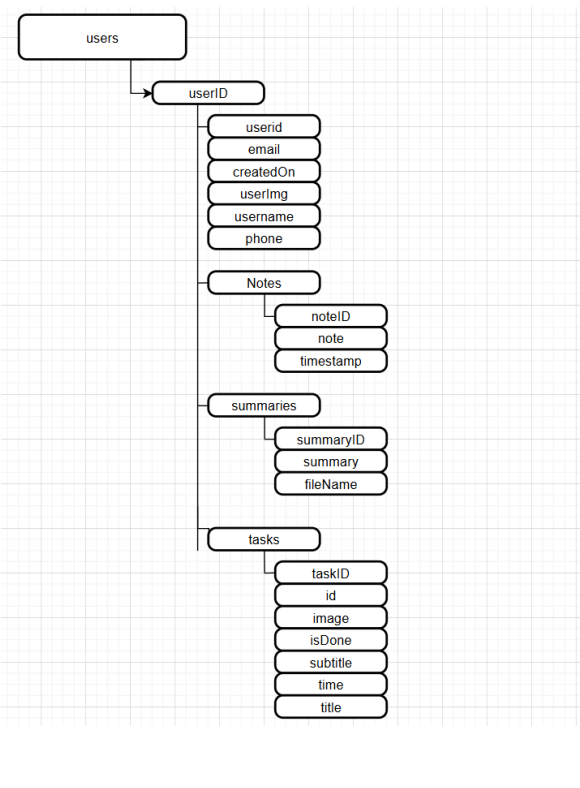
* Firebase stores data in documents (Firestore) or JSON trees (Realtime Database).

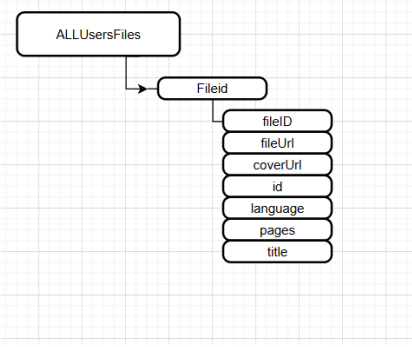
• Data is stored in key-value pairs, with a flexible structure.

• Firebase support: Flexible schema doesn’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 stores and synchronizes 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, to-do list)**.**

### **10.3 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.

## **11. Implementation**

### **11.1 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.

### **11.2 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 developed using Flask to enable communication between front-end and back-end systems. Error logging and handling were properly done to improve reliability and ease debugging. AI features were introduced through the Google Gemini API, which provided sufficient text summarization and chatbot functionality. API requests and responses were handled with caution by the development team to keep them efficient. Firebase was used for storing data in real-time using Firestore, and Firebase Authentication ensured secure user access.

Finally, there was a lot of testing, including unit tests for each and every component and integration tests to verify that the frontend, the backend, and the AI services collaborated together perfectly. This was necessary in delivering a stable and quality application.

### **11.3 Validation**

The validation process ensured that the application met functional and non-functional needs. Several critical tests were conducted to ensure performance, security, and usability. **Functional Testing** was focused on significant functionalities like PDF upload, summarization using AI, chatbot response using AI, and the note-taking feature. **Test documents** were used to ensure the accuracy and reliability of summaries produced by AI. **Performance Testing** ensured the app's response on different devices. **Security Testing** ensured Firebase Authentication operated as expected, protecting users' data and blocking unauthorized access. The results of the testing ensured the application operated as designed, meeting all design requirements and having a glitch-free user interface.

## **12. Results and Discussion**

### **12.1 Summary of Goals Met by the Design**

The application served its main purposes very well, being a pragmatic and sturdy digital document management system. Through the implementation of the Google Gemini API, the application gives quick and precise AI-based summarization, hugely boosting productivity. With the presence of the chatbot, the usability is improved even further with simple, easy-to-understand answers to user questions, which makes the usage of documents an even more productive and natural process. Document management activities were at the forefront, with PDF uploading, reading, and searching being seamless. The app supports searchable PDFs, which allow users to easily search and highlight important words in their documents. This feature makes information more accessible, saving users' time and effort. For extra productivity, the app includes note-taking and to-do lists features, all in real-time synchronization using Firebase. The "Continue Reading" and bookmarking functions enhance usability through simple navigation and document progress tracking.

Security and data integrity were among the top priority considerations utilizing Firebase Authentication, protecting user accounts, and Cloud Firestore, providing durable persistent real-time data integrity. Combined, they supply a secure and stable environment to handle confidential documents.

In general, the app combines new AI capabilities with productive features within a cross-platform application. Future releases will be aimed at advancing AI capability and further optimizing performance to meet users' evolving demands. The successful implementation of such capability validates the effectiveness of the app as a whole document management system.

## **13. 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 a 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 improve the application's usability.

From user testing and feedback, the app was found to be effective in reducing content consumption complexity and enhancing productivity. There is always room for improvement, though. Future enhancements can be in the form of extended language support for translation, optimizing chatbot output, and offline functionality.

In short, the project was able to achieve what it was designed to do through the provision of an effective and accessible digital content management platform. It showcases the ability of AI to enhance the digital experience and opens new fronts for e-reading innovation.

## **14. Recommendations**

Future Improvements and Expansion Recommendations based on the experience and knowledge gathered during the piloting and development of the AI-Based Interactive Digital Content Application, the following are the future improvements and expansion recommendations:

1. Enhance AI Capabilities: Enhance the accuracy and context sensitivity of the chatbot and AI-powered summarization capabilities. More advanced natural language processing (NLP) models can enhance the user experience.

2. Expand Language Support: Bring the translation widget to more languages to allow the app to be used by a global user base.

3. Offline Mode: Add offline functionality for some of the functionalities, such as reading previously summarized content, saved notes, and PDFs. This will make it more useful where internet access is poor.

4. Advanced Search and Filter Options: Add more advanced search capabilities, such as keyword suggestions and voice search, to enable effortless content discovery.

5. Customization of user interface: Provide users with an option to adjust the look and feel, text size, reading mode, and options for text layouts for a more comfortable read.

6. Platform Support for all devices: Make desktop and web versions as well, with equal user interaction experience across platforms.

7. More Security Measures: Implement greater security measures, including 2-factor authentication and advanced encryption, to secure customer data.

8. User Feedback Integration: Develop a feedback mechanism within the application, through which users can input bug reports as well as propose ideas for forthcoming updates.

9. Video summarization: Integrate users in summarizing videos as well.

If these tips are implemented, the application shall be strengthened, interactive, and user-friendly, thus driving wider use and satisfaction among consumers.

## **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: 8 GB (minimum) / 16GB (recommended for smoother performance)

- Storage: 10 GB 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 the supervisor's feedback. Clear action items were included in each stage to guarantee consistent progress toward project handover.

## **User Manual**

Apk installation, Click this [**APK LAN**](https://drive.google.com/file/d/1J3eStYxQYwzUdiCsj5orQV8EymD6IT-a/view?usp=drive_link)

This app is just for Android phones. click this [**User Manual**](https://docs.google.com/presentation/d/1y1PU87xt3TMSUuNilBhLskxLSFJMMiOt/edit?usp=drive_link&ouid=105608814997717740089&rtpof=true&sd=true)