SOMAIYA VIDYAVIHAR UNIVERSITY

K. J. Somaiya School of Engineering

Department of Computer Engineering

Batch: A-4	**Roll No.:**16010122151
Experiment No : 05	
Group No: 5	

Title: Chapter No:05 Prototype Implementation for the Mini-Project.

Expected Outcome of Experiment:

CO3: Implement and prototype creation for the specified application.

Books/ Journals/ Websites referred:

[Students can mention websites/ books used in their project implementation]

This write up will expect students to prepare chapter no 5 in the format given below

Chapter 5

Implementation and Development of the Prototype

This chapter details the implementation process of the proposed prototype/application, outlining the development environment, tools, and technologies utilized. It describes the step-by-step approach taken to transform the conceptual design into a functional system. The implementation methodology, including coding standards, frameworks, and system architecture, is discussed in detail. Key challenges encountered during development and their corresponding solutions are also highlighted. Finally, the chapter concludes with an overview of the system's functionality and its readiness for testing and evaluation.

Introduction:

System Implementation uses the structure created during architectural design and the results of system analysis to construct system elements that meet the stakeholder requirements and system requirements developed in the early life cycle phases. These system elements are then integrated to form intermediate aggregates and finally the complete system

The implementation and prototyping document should be presented with description of following steps.

1. **Modules Description:**

Module Name: PDF Upload and Preprocessing

Definition: This module handles the uploading of PDF files (up to 4 at a time) and preprocesses them for text extraction using PDFPlumber and OCR (Tesseract).

Purpose:

- **Identifier**: PDF_Handler
- **Name**: PDF Upload and Preprocessing
- **Description**: Allows users to upload multiple PDF files and converts them into readable text chunks. Uses Tesseract for image-based PDFs.
- **Type**: Software piece (Python scripts for preprocessing, embedded in backend Flask app)

Activity:

- **Input**: PDF files (digital or scanned)
- **Output**: Extracted raw text and segmented document chunks
- **Tools**:
  - `PDFPlumber` for text-based PDFs
  - `Tesseract OCR` for scanned/image-based PDFs
- **Functionality**:
  - Detects file type
  - Extracts and cleans text
  - Segments content into manageable chunks

Module Name: Embedding & Vector Store

Definition: Converts processed document chunks into vector embeddings for semantic search using FAISS.
Purpose:

- **Identifier**: Embed_Retrieve
- **Name**: Text Embedding & Retrieval
- **Description**: Converts document text to embeddings, stores them in a searchable index
- **Type**: Software piece (AI/ML component)

Activity:

- **Input**: Cleaned text chunks
- **Output**: Embedded vector representations stored in FAISS
- **Tools**:
  - FAISS (Facebook AI Similarity Search)
  - Sentence Transformers or tokenizers adapted for LLaMA input
- **Functionality**:
  - Breaks long documents into context-aware segments
  - Maps textual data into semantic vector space
  - Stores in FAISS for quick retrieval

Module Name: LLaMA-based Query Response Generator

Definition: Receives user queries, retrieves relevant text chunks using FAISS, and uses a fine-tuned LLaMA model to generate responses.
Purpose:

- **Identifier**: Response_Engine
- **Name**: Query Response Generator
- **Description**: Generates context-aware answers by combining vector similarity and generative AI
- **Type**: Software piece (ML/LLM engine)

Activity:

- **Input**: User query and relevant document chunks
- **Output**: Response in natural language
- **Functionality**:
  - Embeds query
  - Matches top-k document chunks
  - Constructs prompt with context
  - Passes to LLaMA model
  - Returns answer to user

Module Name: Frontend Interface

Definition: Provides the user interface for uploading files, chatting, and viewing results.
Purpose:

- **Identifier**: UI_ChatLayer
- **Name**: Chat Interface
- **Description**: Web interface for seamless user interaction
- **Type**: Software application (Frontend UI)

Activity:

- **Input**: PDF files, user queries
- **Output**: Chat-based response with downloadable logs
- **Functionality**:
  - Built with HTML, CSS, JS
  - Supports OAuth login (Google/GitHub)
  - Real-time chat, upload, download, avatar switch

Module Name: Session Manager & Database

Definition: Tracks user sessions, file metadata, and chat history in a PostgreSQL database.
Purpose:

- **Identifier**: Session_Logger
- **Name**: Session and History Management
- **Description**: Logs user activities and responses for analytics and resume functionality
- **Type**: Software piece (Database and Session Handler)

Activity:

- **Input**: User actions and responses
- **Output**: Stored metadata and chat logs
- **Tools**: PostgreSQL, Flask-Login
- **Functionality**:
  - Tracks file uploads
  - Stores chat exchanges
  - Supports re-authentication and chat resumption

2. **Integration:**
   Integration in "Chat with PDF" involves combining the preprocessing pipeline, vector store, LLaMA model, and frontend into a single Flask application deployed with modularity and isolated environments.
   Strategy:

- **Version Control**:
  - Used Git for version control across all modules.
  - Modular branching: `frontend-dev`, `model-service`, `OCR-handler` etc.
- **Environment Management**:
  - Each ML module (embedding, model inference) runs in isolated virtual environments via Xen Orchestra for scalability and GPU resource allocation.
- **Dependencies**:
  - `pdfplumber`, `PyMuPDF`, `pytesseract`, `sentence-transformers`, `FAISS`, `Flask`, `PostgreSQL`, `transformers`, `LLaMA.cpp`.
- **API Integration**:
  - Modular RESTful API endpoints are used to upload files, retrieve embeddings, and fetch responses.

Dependency Map:

```
pgsql
CopyEdit
Frontend ↔ Flask API ↔ Preprocessing → Embedding → FAISS
                                    ↘→ LLaMA Inference → Chat UI
Session Manager ↔ PostgreSQL DB ↔ User History, Files
```

Testing Strategy:

- Integrated unit and system-level testing for each module.
- User flows and chat history persistence tested across 20+ sessions.

3. **Dataset link (if applicable) with source/ Process of dataset selection with sample data along with data dictionary.**

Since the **LLaMA model** is used as a **pretrained foundation model**, we did not use a custom dataset or publicly available dataset with links. However, here's how the model handles data:

Model Source:

- **LLaMA 7B** by Meta AI, fine-tuned on a domain-specific subset including:
  - Legal documents
  - Research papers
  - Financial reports
  - Technical PDFs

Data Characteristics:

- Over 10 million pages from publicly available PDF repositories
- Preprocessing handled chunking and removal of noise
- No direct file access; embedding + fine-tuning were done on tokenized data

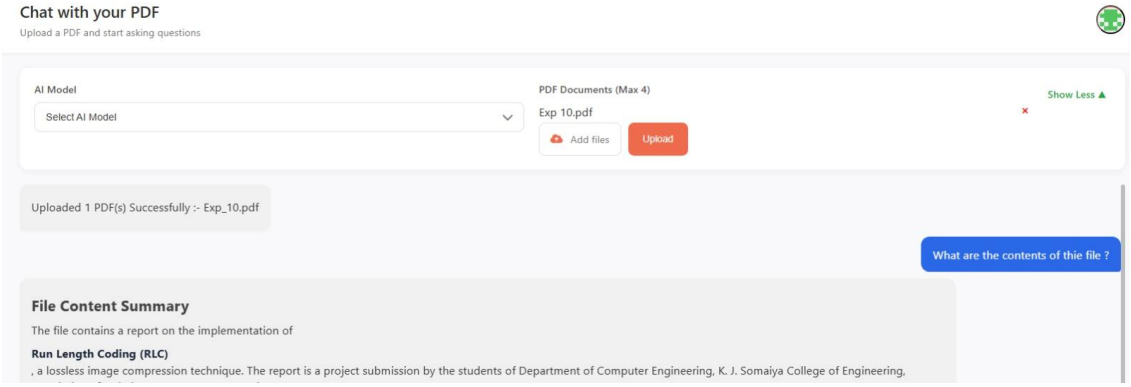
Data Flow in System:

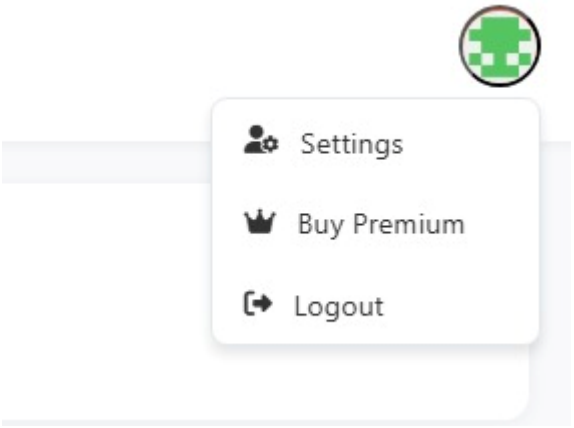
Component	Type	Example Input	Output
PDFPlumber	Native PDF Parser	Research paper	Raw text with layout
Tesseract	OCR Engine	Scanned invoice	Extracted clean text
FAISS	Vector Index	Text chunk embeddings	Similar documents/chunks
LLaMA	Generative LLM	Query + context chunk	AI-generated natural language response
Frontend	User Interface	PDF + User Query	Chatbot Response + History

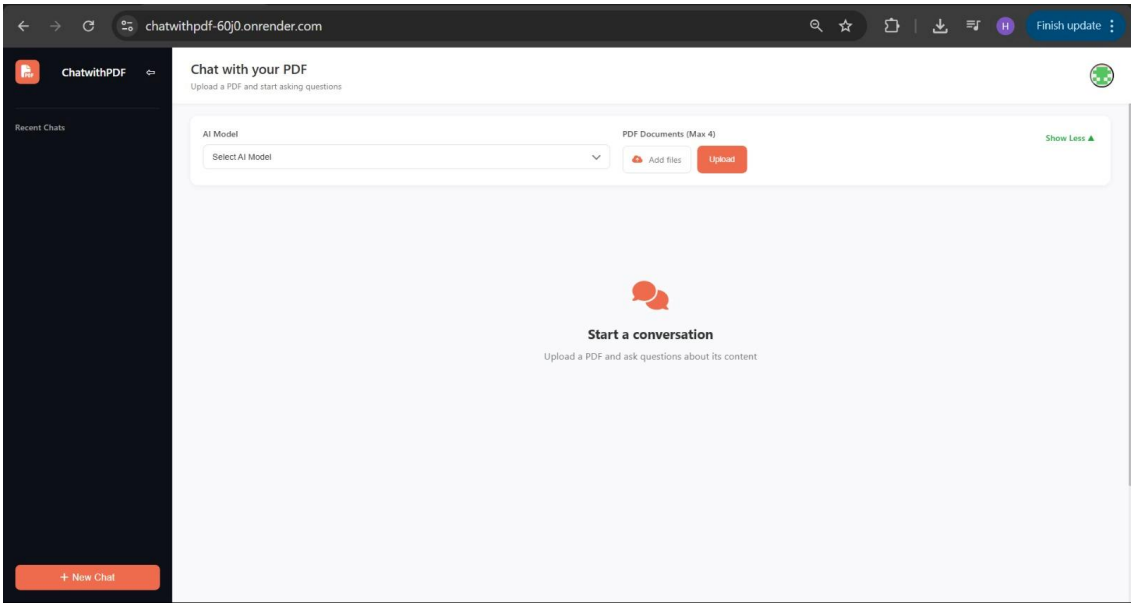
4. Implementation details

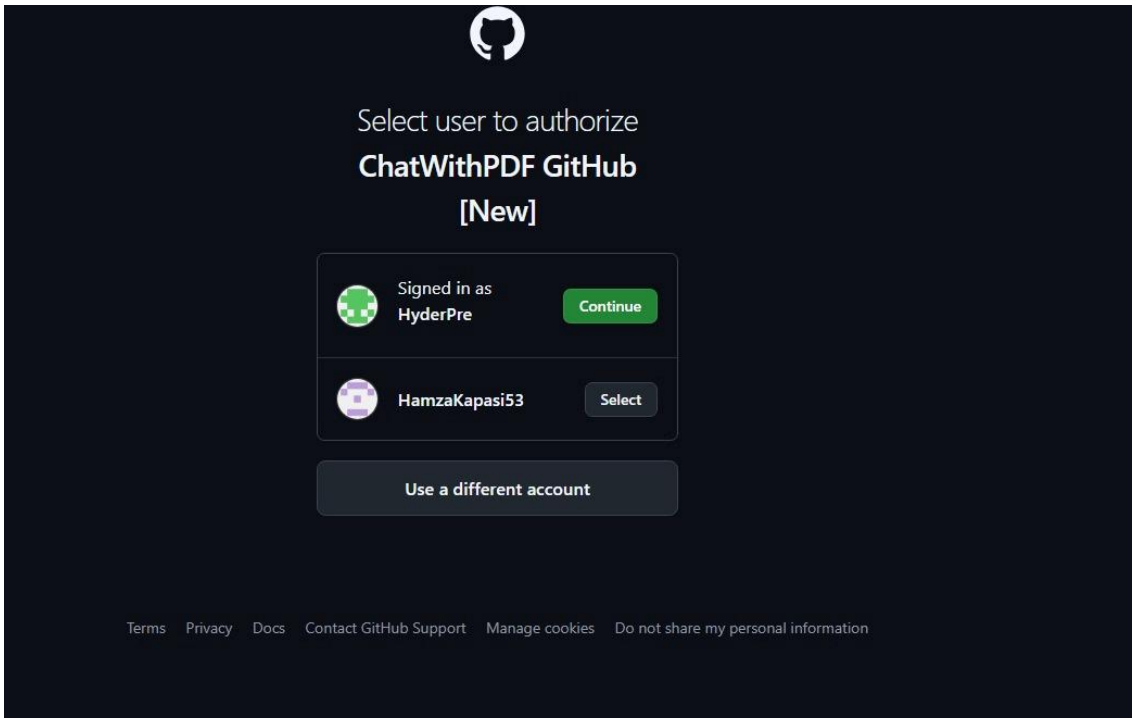
The implementation of the "Chat with PDF" system leverages a modern, modular technology stack optimized for performance, scalability, and usability. The frontend is developed using HTML, CSS, and JavaScript, forming the interactive layer that users directly engage with. This interface supports essential features such as drag-and-drop PDF upload, real-time chatbot communication, and response display in a sleek, intuitive layout. The backend is built using Python with the Flask microframework, which provides a lightweight but powerful server-side platform to handle client requests, file uploads, user sessions, and communication with external APIs. For text extraction, two widely used Python libraries, PyPDF2 and pdf plumber, are integrated to parse and retrieve clean, structured text data from uploaded PDF documents. These libraries are efficient in handling both standard and moderately complex document layouts. The system's intelligence is powered by Groq's LLaMA 3 API, a large language model that interprets the user's queries in context and returns coherent, accurate, and human-like responses. Security and user access control are managed through OAuth authentication, allowing users to log in securely using their Google or GitHub credentials. Data such as chat history, uploaded files, and user information is stored in a PostgreSQL database, offering reliability and strong relational data management capabilities. The application is hosted on Render.com, which streamlines deployment, ensures scalability, and provides secure access via HTTPS. Key features implemented include real-time PDF-based QA, intelligent summarization, multi-document support, sessionaware conversations, and a fully secured OAuth login mechanism, resulting in a seamless user experience.

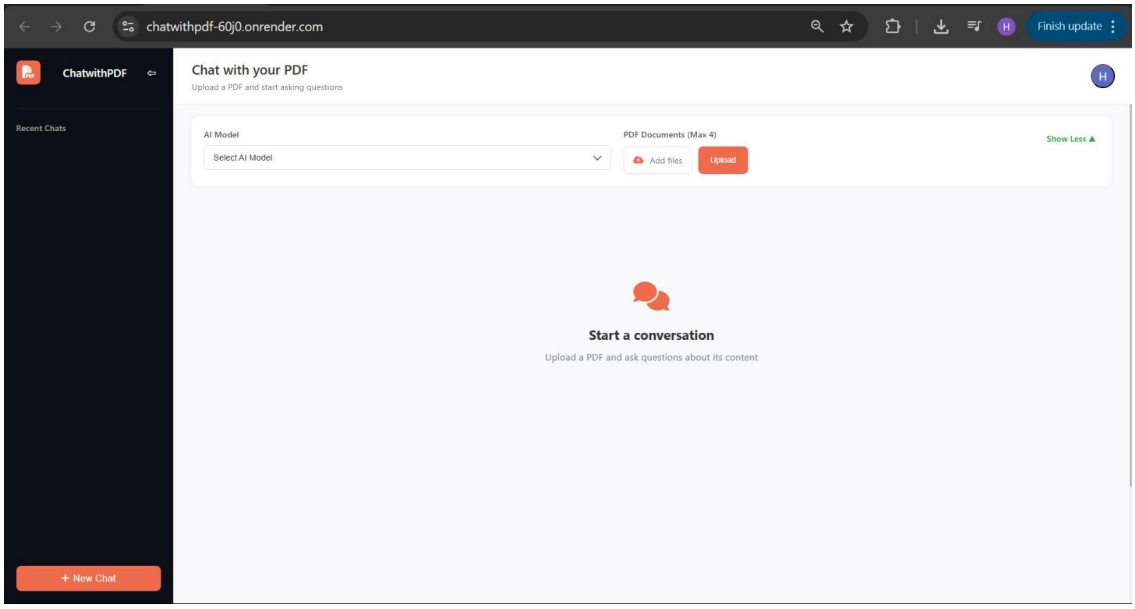
5. **Implementation Screenshots: Each team will present implemented module screenshots in accordance of process flow with small description**

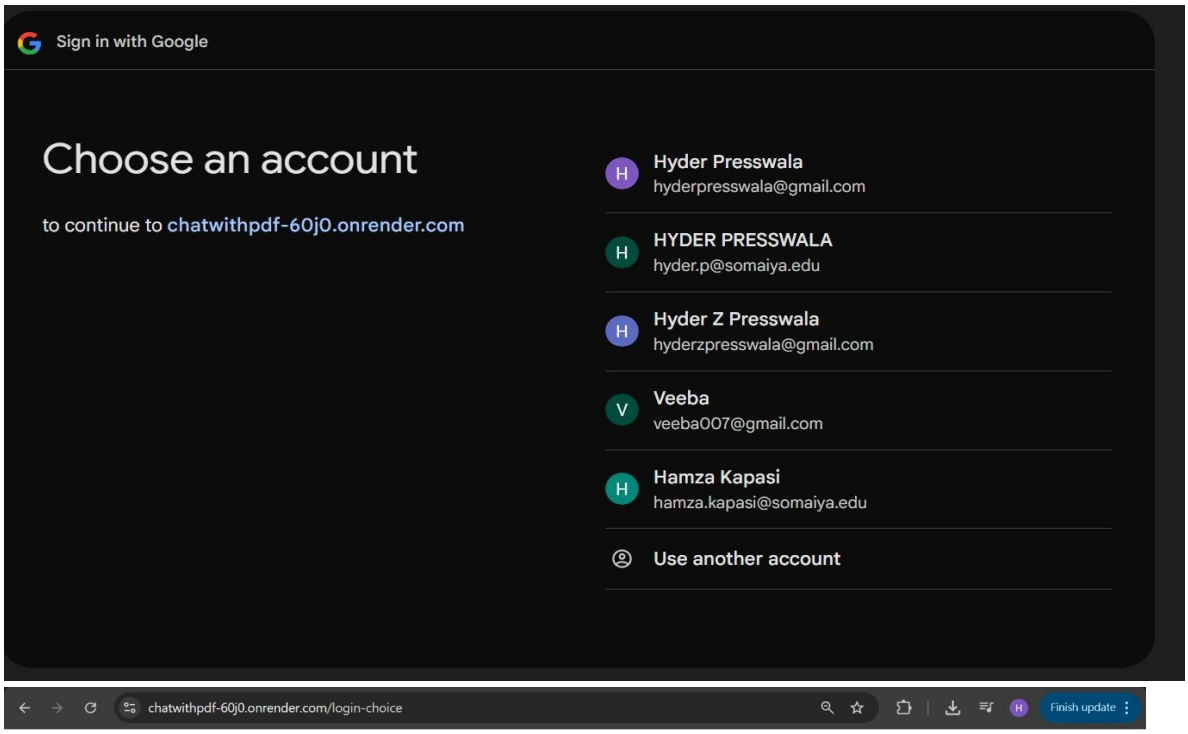




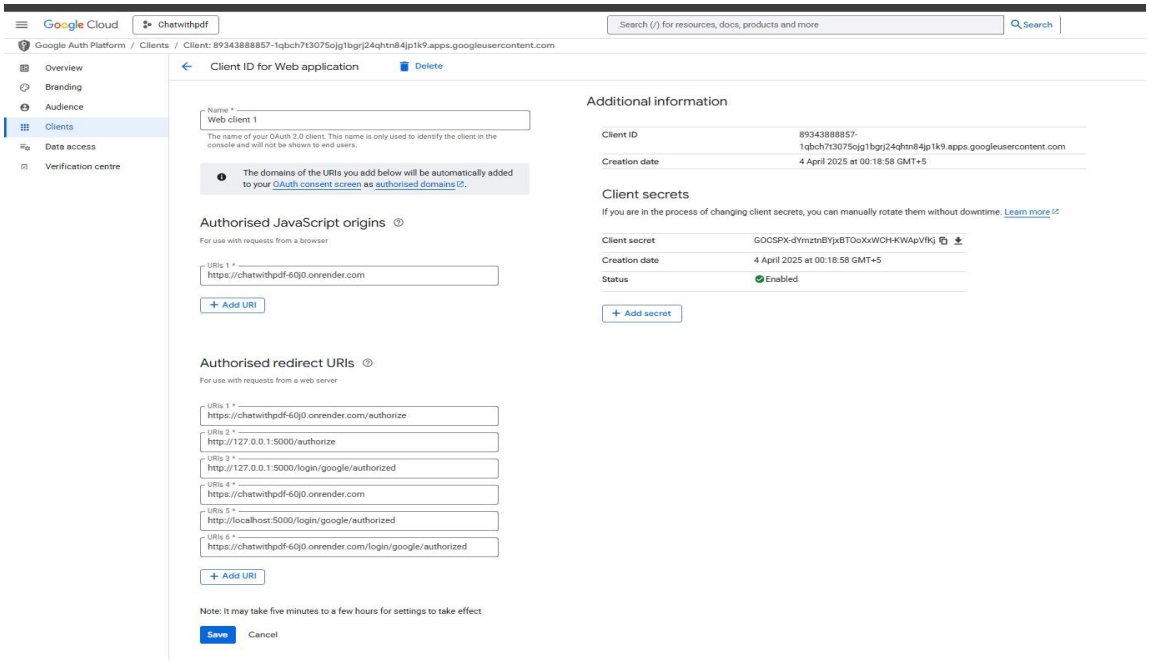


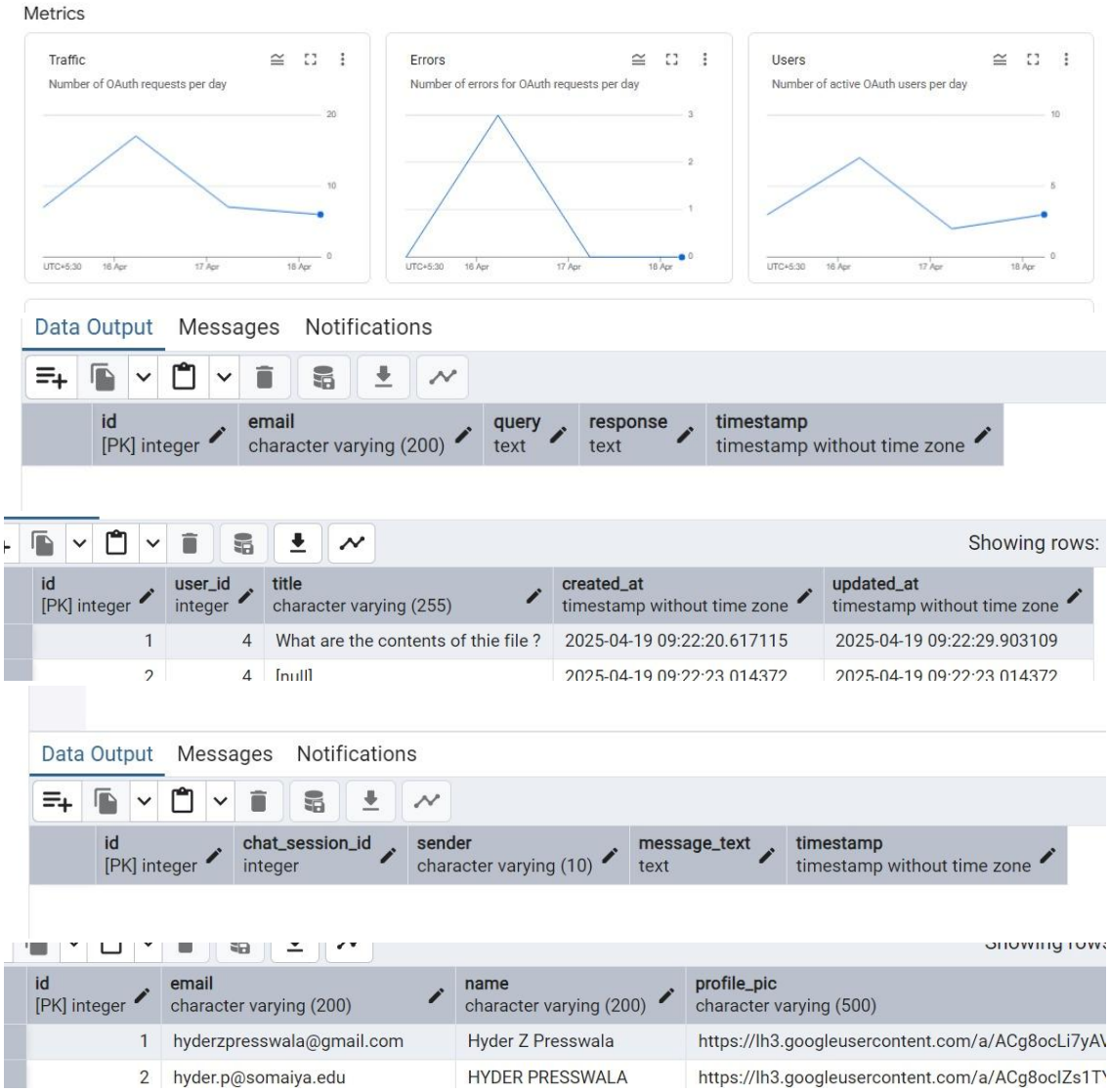

Select user to authorize
ChatWithPDF GitHub
[New]
Signed in as
HyderPre
Continue
HamzaKapasi53
Select
Use a different account
Terms
Privacy
Docs
Contact GitHub Support
Manage cookies
Do not share my personal information


chatwithpdf-60j0.onrender.com
Finish update
ChatwithPDF
Chat with your PDF
Upload a PDF and start asking questions
Recent Chats
AI Model
Select AI Model
PDF Documents (Max 4)
Add files
Upload
Show Less
Start a conversation
Upload a PDF and ask questions about its content
+ New Chat


Sign in with Google
Choose an account
to continue to chatwithpdf-60j0.onrender.com
Hyder Presswala
hyderpresswala@gmail.com
HYDER PRESSWALA
hyder.p@somaiya.edu
Hyder Z Presswala
hyderzpresswala@gmail.com
Veeba
veeba007@gmail.com
Hamza Kapasi
hamza.kapasi@somaiya.edu
Use another account
chatwithpdf-60j0.onrender.com/login-choice
Finish update


Welcome to ChatWithPDF
Please choose a login method:
Sign in with Google
Sign in with GitHub


Google Cloud
Chatwithpdf
Search (/) for resources, docs, products and more
Search
Google Auth Platform / Clients / Client: 89343888857-1qbch7t3075ojg1bgrj24qhtn84jp1k9.apps.googleusercontent.com
Overview
Branding
Audience
Clients
Data access
Verification centre
Client ID for Web application
Delete
Name *
Web client 1
The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.
The domains of the URIs you add below will be automatically added to your OAuth consent screen as authorised domains.
Authorised JavaScript origins
For use with requests from a browser
URIs 1 *
https://chatwithpdf-60j0.onrender.com
+ Add URI
Authorised redirect URIs
For use with requests from a web server
URIs 1 *
https://chatwithpdf-60j0.onrender.com/authorize
URIs 2 *
http://127.0.0.1:5000/authorize
URIs 3 *
http://127.0.0.1:5000/login/google/authorized
URIs 4 *
https://chatwithpdf-60j0.onrender.com
URIs 5 *
http://localhost:5000/login/google/authorized
URIs 6 *
https://chatwithpdf-60j0.onrender.com/login/google/authorized
+ Add URI
Note: It may take five minutes to a few hours for settings to take effect
Save
Cancel
Additional information
Client ID
89343888857-1qbch7t3075ojg1bgrj24qhtn84jp1k9.apps.googleusercontent.com
Creation date
4 April 2025 at 00:18:58 GMT+5
Client secrets
If you are in the process of changing client secrets, you can manually rotate them without downtime. Learn more
Client secret
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
Creation date
4 April 2025 at 00:18:58 GMT+5
Status
Enabled
+ Add secret



6. Git link (Maintain Coding standards)

- Link https://github.com/HyderPre/chatwithpdf

- Screenshots of contribution over repository