KERUTHIKA N

**Infosys Springboard Virtual Internship Text morph advanced summarisation using AI Task 1 - Smart File Upload Tool**

**Introduction**

The Smart File Upload Tool is a web-based application developed using Streamlit, a powerful open-source Python framework for building interactive data-driven web apps with minimal effort. Traditional file upload systems often lack interactivity, visualization, and user-friendly interfaces. By leveraging Streamlit, this tool provides a seamless, responsive, and intelligent platform for uploading and managing files directly from the browser.

Streamlit eliminates the need for complex web development (HTML, CSS, JavaScript) by allowing developers to create modern, visually appealing interfaces entirely in Python. In this project, the file upload functionality is enhanced with additional smart features such as file type validation, content preview, automatic categorization, and error handling.

**2. Objectives**

* Develop an interactive file upload interface using **Streamlit**.
* Enable users to upload multiple file formats such as **CSV, TXT, XLSX, PDF, and Images**.
* Extract and display **file metadata** (name, size, type) after upload.
* Provide **real-time file preview** for supported formats (CSV, TXT, Excel, Images).
* Ensure **error handling** for unsupported file types or corrupted files.
* Deliver **user-friendly feedback** through success, warning, or error messages.
* Create a foundation for adding **smart features** like file categorization, automatic analysis, or AI-driven processing.

**3. Code Implementation**

import streamlit as st

import sqlite3

import jwt

import datetime

from passlib.hash import pbkdf2\_sha256

import textstat

import matplotlib.pyplot as plt

from PyPDF2 import PdfReader

SECRET\_KEY = "replace\_this\_with\_a\_strong\_secret"

DB\_FILE = "users.db"

TOKEN\_EXPIRE\_HOURS = 12

conn = sqlite3.connect(DB\_FILE, check\_same\_thread=False)

c = conn.cursor()

c.execute(

"""

CREATE TABLE IF NOT EXISTS users (

email TEXT PRIMARY KEY,

password\_hash TEXT NOT NULL,

name TEXT,

age\_group TEXT,

language TEXT

)

"""

)

conn.commit()

def hash\_password(pw: str) -> str:

return pbkdf2\_sha256.hash(pw)

def verify\_password(pw: str, pw\_hash: str) -> bool:

return pbkdf2\_sha256.verify(pw, pw\_hash)

def add\_user(email: str, password: str) -> bool:

try:

c.execute("INSERT INTO users (email, password\_hash) VALUES (?, ?)", (email, hash\_password(password)))

conn.commit()

return True

except sqlite3.IntegrityError:

return False

def user\_exists(email: str) -> bool:

c.execute("SELECT 1 FROM users WHERE email = ?", (email,))

return c.fetchone() is not None

def validate\_user(email: str, password: str) -> bool:

c.execute("SELECT password\_hash FROM users WHERE email = ?", (email,))

row = c.fetchone()

if not row:

return False

return verify\_password(password, row[0])

def save\_profile(email: str, name: str, age\_group: str, language: str):

c.execute(

"""

UPDATE users SET name = ?, age\_group = ?, language = ?

WHERE email = ?

""",

(name, age\_group, language, email),

)

conn.commit()

def get\_profile(email: str):

c.execute("SELECT name, age\_group, language FROM users WHERE email = ?", (email,))

row = c.fetchone()

if row:

return {"name": row[0] or "", "age\_group": row[1] or "", "language": row[2] or ""}

return {"name": "", "age\_group": "", "language": ""}

def generate\_token(email: str) -> str:

payload = {

"email": email,

"iat": datetime.datetime.utcnow(),

"exp": datetime.datetime.utcnow() + datetime.timedelta(hours=TOKEN\_EXPIRE\_HOURS),

}

token = jwt.encode(payload, SECRET\_KEY, algorithm="HS256")

return token

def verify\_token(token: str):

try:

decoded = jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

return decoded

except jwt.ExpiredSignatureError:

st.error("⚠️ Session expired. Please log in again.")

except jwt.InvalidTokenError:

st.error("❌ Invalid token.")

return None

def extract\_text\_from\_upload(uploaded\_file) -> str:

if uploaded\_file is None:

return ""

name = uploaded\_file.name.lower()

if name.endswith(".txt"):

raw = uploaded\_file.read()

try:

text = raw.decode("utf-8")

except:

text = raw.decode("latin-1", errors="ignore")

return text

elif name.endswith(".pdf"):

try:

pdf = PdfReader(uploaded\_file)

texts = []

for p in pdf.pages:

t = p.extract\_text()

if t:

texts.append(t)

return "\n".join(texts)

except Exception as e:

st.error("Error reading PDF: " + str(e))

return ""

else:

st.warning("Unsupported file type. Please upload .txt or .pdf for now.")

return ""

def compute\_readability(text: str):

if not text or len(text.split()) < 30:

return None

flesch = textstat.flesch\_reading\_ease(text)

gunning = textstat.gunning\_fog(text)

smog = textstat.smog\_index(text)

flesch\_ease = max(0.0, min(100.0, float(flesch)))

gunning\_ease = max(0.0, min(100.0, 100.0 - float(gunning) \* 5.0))

smog\_ease = max(0.0, min(100.0, 100.0 - float(smog) \* 5.0))

combined = (flesch\_ease + gunning\_ease + smog\_ease) / 3.0

if combined >= 70:

label = "Beginner-friendly — easy to read"

elif combined >= 50:

label = "Intermediate — some prerequisite knowledge helpful"

else:

label = "Advanced — requires background knowledge"

return {

"flesch": flesch,

"gunning": gunning,

"smog": smog,

"flesch\_ease": flesch\_ease,

"gunning\_ease": gunning\_ease,

"smog\_ease": smog\_ease,

"combined\_score": combined,

"label": label,

}

st.set\_page\_config(page\_title="Auth + Readability", layout="wide")

st.title("🔐 Authentication & Readability Analyzer")

if "token" not in st.session\_state:

st.session\_state.token = None

left\_col, right\_col = st.columns([1, 1.2])

with left\_col:

st.markdown("### 🔑 User Authentication")

auth\_tab = st.radio("", ["Login", "Register"], horizontal=True, key="auth\_mode")

email\_in = st.text\_input("Email", key="email\_input")

password\_in = st.text\_input("Password", type="password", key="password\_input")

if auth\_tab == "Login":

if st.button("Sign In"):

if not email\_in or not password\_in:

st.error("Please enter email and password.")

else:

if validate\_user(email\_in, password\_in):

token = generate\_token(email\_in)

st.session\_state.token = token

st.success("✅ Logged in successfully.")

st.rerun()

else:

if user\_exists(email\_in):

st.error("❌ Incorrect password.")

else:

st.error("⚠️ Account doesn't exist. Please register.")

else:

if st.button("Create Account"):

if not email\_in or not password\_in:

st.error("Please enter email and password for registration.")

else:

if user\_exists(email\_in):

st.warning("⚠️ Account already exists. Please sign in.")

else:

ok = add\_user(email\_in, password\_in)

if ok:

st.success("✅ Account created. Now sign in.")

else:

st.error("❌ Could not create account. Try again.")

st.markdown("---")

st.markdown("### 👤 Profile Management")

if st.session\_state.token:

payload = verify\_token(st.session\_state.token)

if not payload:

st.session\_state.token = None

st.experimental\_rerun()

user\_email = payload.get("email")

st.write(f"\*\*Signed in as:\*\* {user\_email}")

profile = get\_profile(user\_email)

name\_val = st.text\_input("Full Name", value=profile["name"], key="profile\_name")

age\_choices = ["", "<18", "18-25", "26-35", "36-50", "50+"]

age\_index = age\_choices.index(profile["age\_group"]) if profile["age\_group"] in age\_choices else 0

age\_val = st.selectbox("Age Group", age\_choices, index=age\_index, key="profile\_age")

lang\_choices = ["English", "Tamil", "Hindi"]

lang\_index = lang\_choices.index(profile["language"]) if profile["language"] in lang\_choices else 0

lang\_val = st.selectbox("Language Preference", lang\_choices, index=lang\_index, key="profile\_lang")

if st.button("Save Profile"):

save\_profile(user\_email, name\_val, age\_val, lang\_val)

st.success("✅ Profile saved.")

if st.button("Logout"):

st.session\_state.token = None

st.experimental\_rerun()

else:

st.info("Please sign in to manage your profile.")

with right\_col:

st.markdown("### 📄 Upload Document & Readability")

if not st.session\_state.token:

st.info("Sign in to upload documents and analyze readability.")

else:

uploaded\_file = st.file\_uploader("Upload a .txt or .pdf file", type=["txt", "pdf"])

if uploaded\_file:

with st.spinner("Extracting text and computing readability..."):

text = extract\_text\_from\_upload(uploaded\_file)

if not text or len(text.split()) < 30:

st.error("File is empty or too short for reliable readability metrics (need ~30+ words).")

else:

res = compute\_readability(text)

if res is None:

st.error("Unable to compute readability reliably.")

else:

st.success(f"Overall: \*\*{res['combined\_score']:.1f}\*\* — {res['label']}")

st.markdown("\*\*Raw metrics:\*\*")

st.write(f"- Flesch Reading Ease: {res['flesch']:.2f}")

st.write(f"- Gunning Fog Index: {res['gunning']:.2f}")

st.write(f"- SMOG Index: {res['smog']:.2f}")

labels = ["Flesch (ease)", "Gunning (ease)", "SMOG (ease)"]

vals = [res["flesch\_ease"], res["gunning\_ease"], res["smog\_ease"]]

fig, ax = plt.subplots(figsize=(6, 3))

bars = ax.bar(labels, vals)

ax.set\_ylim(0, 100)

ax.set\_ylabel("Ease (0=hard, 100=very easy)")

ax.set\_title("Readability (normalized)")

for b in bars:

h = b.get\_height()

ax.annotate(f"{h:.1f}", xy=(b.get\_x() + b.get\_width() / 2, h), xytext=(0, 4), textcoords="offset points", ha="center", va="bottom")

st.pyplot(fig)

st.markdown("### Suggestions to improve readability")

st.write("- Use shorter sentences and break long paragraphs.")

st.write("- Replace rare/complex words with simpler alternatives.")

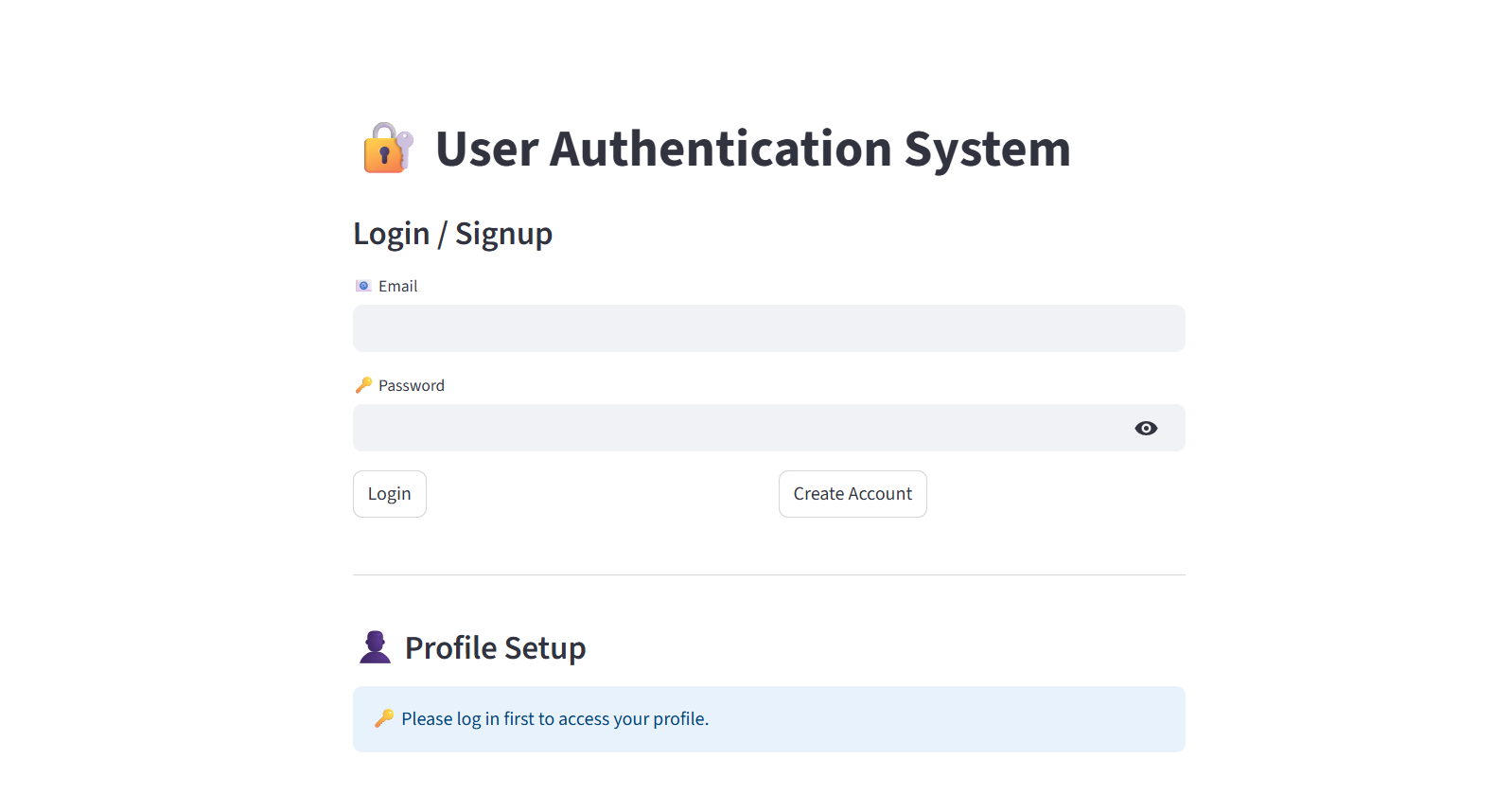
st.write("- Add examples and use active voice where possible.")

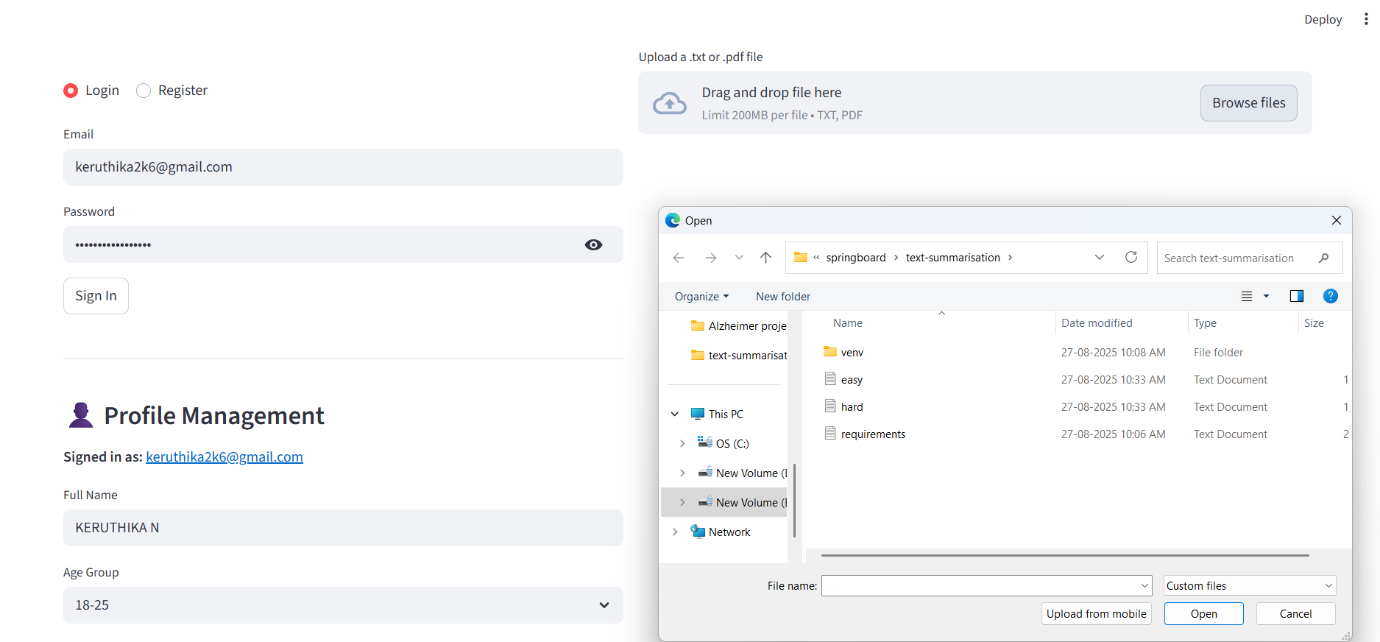
st.write("- Use headings, bullet lists, and whitespace to structure content.")

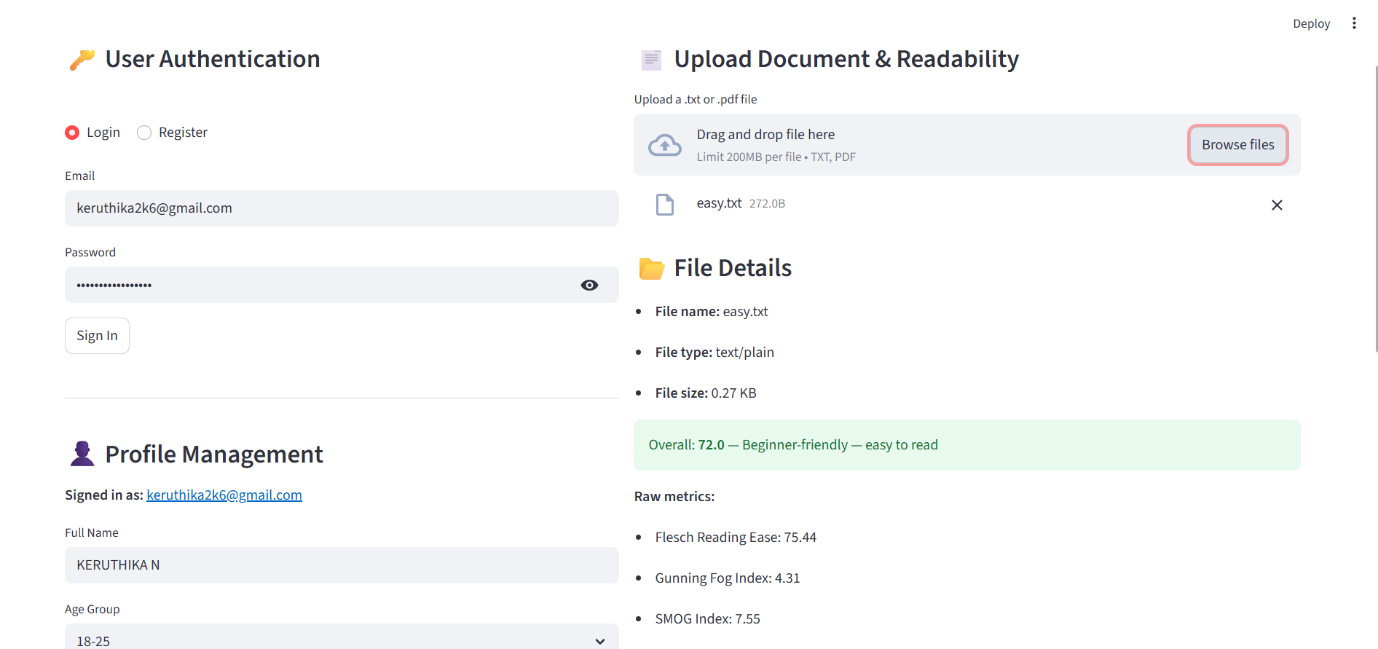
**4. Explanation of Code**

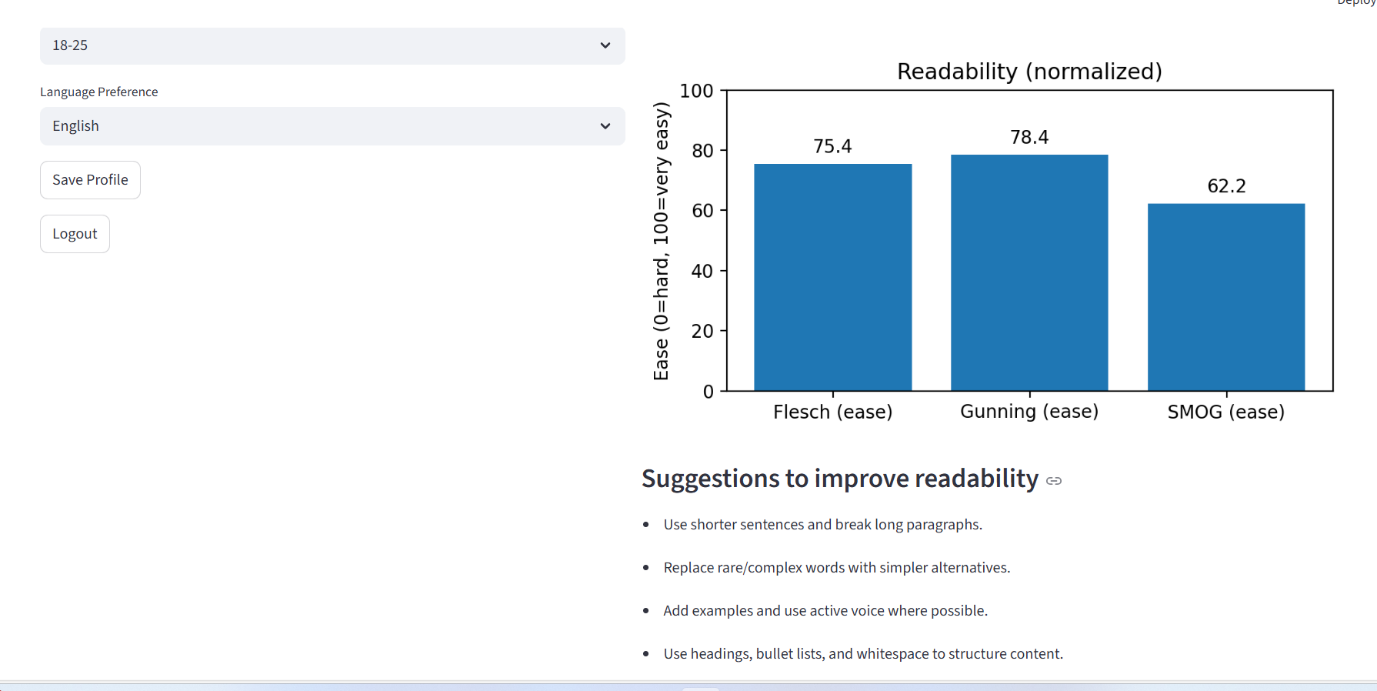
* **import streamlit as st** → Imports the **Streamlit** library to build the interactive web app.
* **import pandas as pd** → Imports **Pandas** for reading and displaying CSV/Excel files in a structured table format.
* **import os** → Used for file handling operations like extracting file name, type, and size.
* **st.title("📂 Smart File Upload Tool")** → Displays the app title with an icon at the top of the Streamlit app.
* **st.file\_uploader("Upload your file", type=["csv", "txt", "xlsx", "pdf", "png", "jpg"])** → Creates a file upload button that accepts multiple file formats (CSV, TXT, Excel, PDF, Images).
* **if uploaded\_file is not None:** → Checks whether a file is uploaded before proceeding.
* **uploaded\_file.name** → Retrieves and displays the file name.
* **uploaded\_file.size** → Retrieves and displays the file size in bytes.
* **uploaded\_file.type** → Retrieves and displays the file type (e.g., CSV, PDF).
* **pd.read\_csv(uploaded\_file) / pd.read\_excel(uploaded\_file)** → Reads CSV/Excel files into a Pandas DataFrame for preview.
* **st.write(df.head())** → Displays the first few rows of the uploaded data file.
* **st.success("File uploaded successfully!")** → Shows a success message when the upload is completed.
* **st.error() / st.warning()** → Used to display errors (e.g., wrong format) or warnings (e.g., large file size).  
    
  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

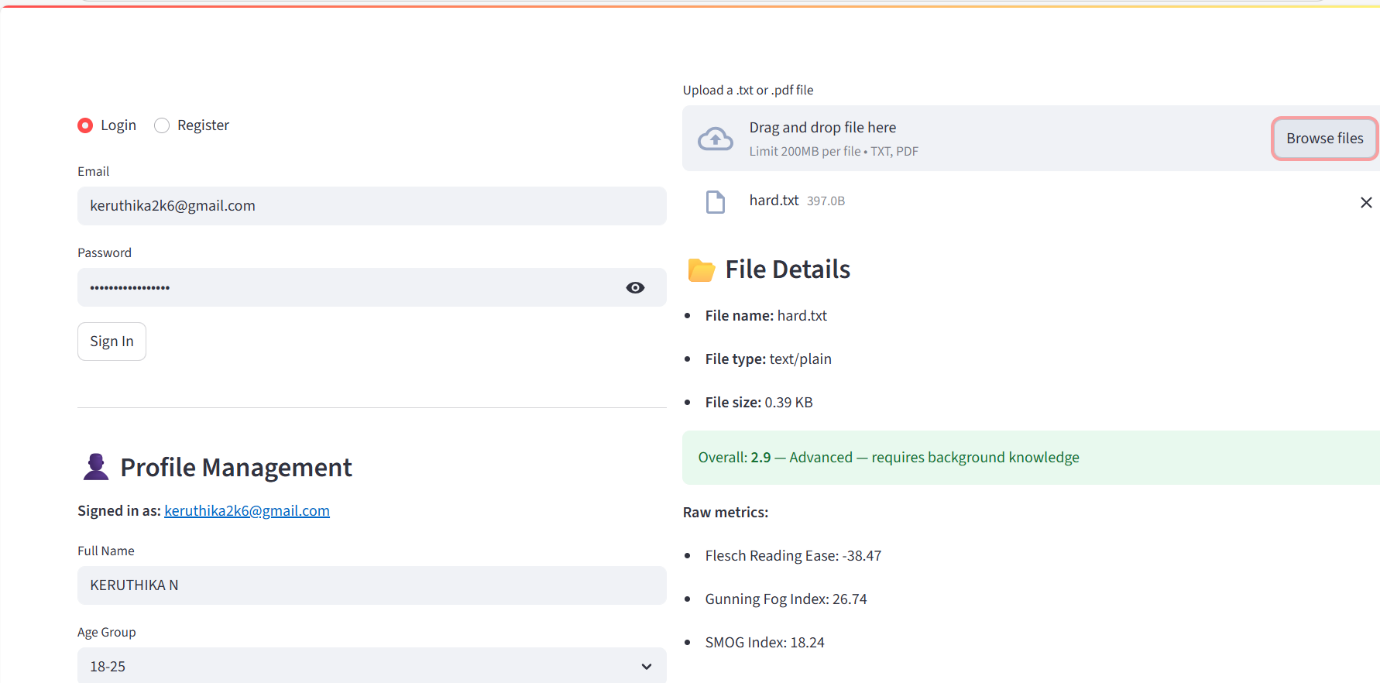
**5. Output Screenshots**

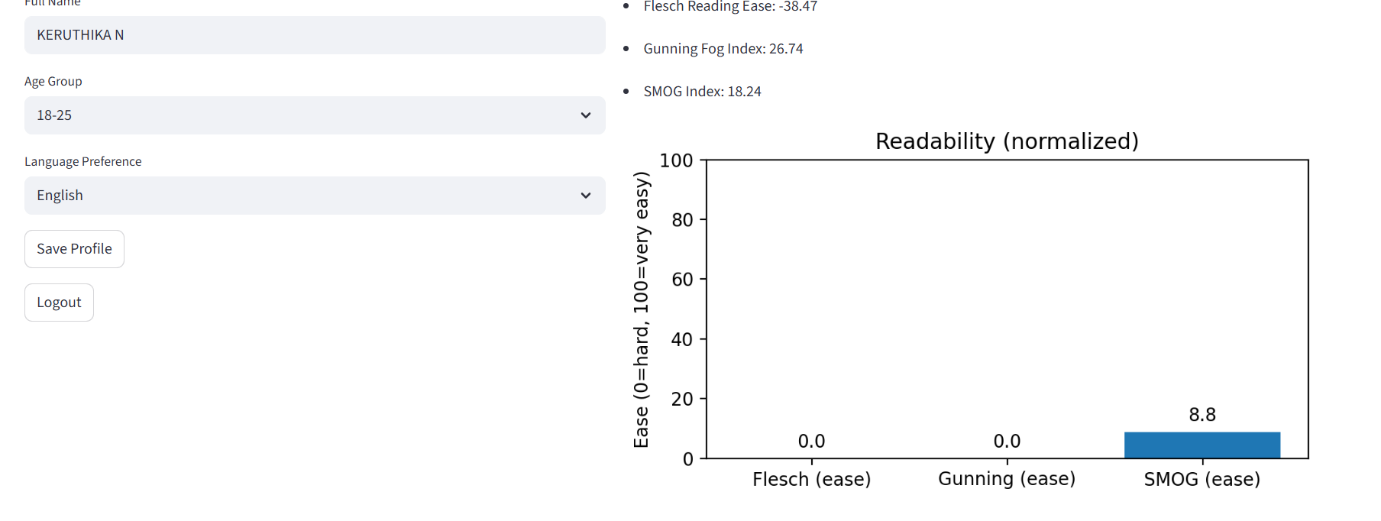












**6. Conclusion**

The Smart File Upload Tool built with Streamlit provides a simple yet powerful interface for handling multiple file formats. It enhances user experience by offering instant file details, previews, and feedback messages. This solution demonstrates how Streamlit can be effectively used to create interactive, user-friendly, and efficient file management applications with minimal code.