**Study Assistant Project Report**

**1. Project Overview**

The **Study Assistant** is a Streamlit-based web application designed to enhance learning through interactive tools and personalized study resources. It targets students by offering features like real-time question answering, study plan generation, quiz creation, flashcard review, progress tracking, and study suggestions. The app’s playful aesthetic, with a black background, pink accents, and Comic Sans font, creates an engaging and motivating user experience.

**Key Features**

1. **Authentication System** (app.py):
   * Secure user signup and login with password hashing (SHA256 with salt).
   * SQLite database stores user credentials and progress data.
2. **Dashboard** (dashboard.py):
   * Centralized navigation hub with a sidebar for accessing all features.
   * Motivational messages and a clean, user-friendly layout.
3. **Real-Time QA Chatbot** (real\_time\_qa.py):
   * Uploads PDF/Word documents and answers questions using LangChain, FAISS, and Groq API.
   * Supports document summarization and conversational history.
4. **Study Plan Generator** (study\_plan\_generator.py):
   * Creates personalized study schedules based on subjects, exam dates, interests, and uploaded materials.
   * Prioritizes topics for upcoming exams.
5. **Quiz Generator** (quiz\_generator.py):
   * Generates quizzes (multiple-choice, true/false, short-answer) from uploaded content or user-specified topics.
   * Tracks and logs quiz performance in the database.
6. **Flashcards Creator** (flashcards.py):
   * Generates flashcards from study materials for interactive review.
   * Logs flashcard review progress.
7. **Progress Tracker** (progress\_tracker.py):
   * Visualizes quiz scores and flashcard reviews with Plotly charts (line and bar).
   * Provides motivational feedback based on performance.
8. **Study Suggestions** (study\_suggestion.py):
   * Recommends study topics based on user progress, prioritizing weak areas.
   * Fetches supplementary resources from Wikipedia.

**2. Technical Architecture**

**Tech Stack**

* **Frontend**: Streamlit for the web interface, with custom CSS for styling.
* **Backend**:
  + **Python Libraries**: LangChain (document processing, embeddings), FAISS (vector storage), SentenceTransformers (custom embeddings), Groq API (LLM), Plotly (visualizations), Pandas (data analysis), SQLite (database).
  + **File Processing**: PyPDFLoader and Docx2txtLoader for PDF/Word document handling.
  + **Web Scraping**: BeautifulSoup for fetching Wikipedia resources.
* **Database**: SQLite for storing user credentials and progress data.
* **Environment**: Managed via .env file for API keys (e.g., GROQ\_API\_KEY).

**File Structure**

* app.py: Main application, handles authentication and page navigation.
* dashboard.py: Central hub for feature navigation.
* real\_time\_qa.py: QA chatbot with document processing.
* study\_plan\_generator.py: Study plan creation.
* quiz\_generator.py: Quiz generation and evaluation.
* flashcards.py: Flashcard creation and review.
* progress\_tracker.py: Progress visualization and feedback.
* study\_suggestion.py: Personalized study recommendations.
* embedding\_generator.py: Currently empty, intended for reusable embedding logic.

**Data Flow**

1. **User Authentication**: Users sign up or log in, with credentials stored in SQLite.
2. **Document Upload**: Users upload PDF/Word files, processed into chunks and stored in a FAISS vector store.
3. **Feature Interaction**:
   * QA, quizzes, and flashcards use the vector store and Groq API to generate content.
   * Study plans and suggestions leverage user inputs and progress data.
4. **Progress Tracking**: Quiz scores and flashcard reviews are logged in SQLite and visualized.
5. **Suggestions**: Progress data informs topic recommendations, enhanced with web resources.

**3. Recent Issues and Resolutions**

**Issue 1: ModuleNotFoundError: No module named 'study\_suggestions'**

* **Description**: The app failed to run due to an incorrect import in dashboard.py, where study\_suggestions (plural) was used instead of study\_suggestion (singular), mismatching the file name.
* **Resolution**:
  + Updated the import in dashboard.py:
  + from study\_suggestion import run\_study\_suggestions
  + Verified the file name (study\_suggestion.py) and function name (run\_study\_suggestions).
  + Ensured the file was in the correct directory and the virtual environment was active.
* **Impact**: Fixed the import error, allowing the app to load the study suggestions feature.

**Issue 2: StreamlitSetPageConfigMustBeFirstCommandError**

* **Description**: The st.set\_page\_config() call in app.py was inside the main() function and preceded by an st.markdown command, violating Streamlit’s requirement that set\_page\_config() be the first command.
* **Resolution**:
  + Moved st.set\_page\_config() to the top of app.py, before any other Streamlit commands:
  + st.set\_page\_config(page\_title="Study Assistant", page\_icon="📚", initial\_sidebar\_state="expanded")
  + Removed the redundant st.set\_page\_config() call from main().
  + Ensured no other Streamlit commands executed before it.
* **Impact**: Resolved the configuration error, enabling the app to run with the correct page settings.

**4. Strengths**

1. **Modular Design**: Each feature is encapsulated in a separate file, enhancing maintainability and scalability.
2. **Engaging UI/UX**: The playful design (black background, pink buttons, Comic Sans font) and motivational messages create a welcoming experience.
3. **Robust Functionality**: Integration of LangChain, FAISS, and Groq API enables powerful document processing and content generation.
4. **Progress Insights**: SQLite and Plotly provide meaningful tracking and visualization of user progress.
5. **Error Handling**: Comprehensive checks for missing API keys, invalid JSON, and user inputs ensure reliability.

**5. Areas for Improvement**

1. **Embedding Consistency**:
   * **Issue**: real\_time\_qa.py uses HuggingFaceEmbeddings, while quiz\_generator.py and flashcards.py use a custom SentenceTransformer class.
   * **Recommendation**: Standardize embeddings using HuggingFaceEmbeddings and move logic to embedding\_generator.py:
   * from langchain\_community.embeddings import HuggingFaceEmbeddings
   * def get\_embeddings():
   * return HuggingFaceEmbeddings(model\_name="sentence-transformers/all-MiniLM-L6-v2")

Update all modules to import and use get\_embeddings().

1. **Vector Store Persistence**:
   * **Issue**: The FAISS vector store in st.session\_state resets on app restart, requiring reprocessing of uploaded files.
   * **Recommendation**: Save the vector store to disk using vector\_store.save\_local() and load it on startup:
   * import os
   * def load\_or\_create\_vector\_store(documents):
   * if os.path.exists("faiss\_index"):
   * return FAISS.load\_local("faiss\_index", get\_embeddings())
   * vector\_store = FAISS.from\_documents(documents, get\_embeddings())
   * vector\_store.save\_local("faiss\_index")
   * return vector\_store
2. **Quiz Answer Evaluation**:
   * **Issue**: Short-answer questions in quiz\_generator.py use basic case-insensitive comparison, which may miss nuanced responses.
   * **Recommendation**: Use the Groq API to evaluate semantic similarity between user and correct answers:
   * def evaluate\_short\_answer(user\_answer, correct\_answer):
   * llm = ChatGroq(groq\_api\_key=os.getenv("GROQ\_API\_KEY"), model\_name="llama3-8b-8192")
   * prompt = f"Are these two answers semantically equivalent?\n1: {user\_answer}\n2: {correct\_answer}\nAnswer: Yes/No"
   * response = llm.invoke(prompt)
   * return response.content.strip() == "Yes"
3. **Session State Management**:
   * **Issue**: Frequent use of st.rerun() (e.g., in flashcards.py) can disrupt user experience.
   * **Recommendation**: Use st.form and callbacks to update state without rerunning the entire app:
   * with st.form("flashcard\_nav"):
   * col1, col2 = st.columns(2)
   * with col1:
   * prev = st.form\_submit\_button("Previous", disabled=st.session\_state.flashcard\_index == 0)
   * with col2:
   * next = st.form\_submit\_button("Next", disabled=st.session\_state.flashcard\_index >= len(st.session\_state.flashcards)-1)
   * if prev:
   * st.session\_state.flashcard\_index -= 1
   * if next:
   * st.session\_state.flashcard\_index += 1
   * log\_progress(st.session\_state.user\_id, topic, flashcards\_reviewed=1)
4. **Resource Fetching**:
   * **Issue**: study\_suggestion.py relies on Wikipedia for resources, which may not always be relevant.
   * **Recommendation**: Expand to include sources like Khan Academy or Coursera using APIs or more robust scraping.

**6. Performance and Scalability**

* **Current Performance**: The app performs well for individual users, with efficient document processing and quick LLM responses via Groq API.
* **Scalability Considerations**:
  + **Database**: SQLite is suitable for small-scale use but may struggle with many users. Consider PostgreSQL for production.
  + **Vector Store**: Persisting FAISS indexes will reduce processing time for repeated uploads.
  + **API Usage**: Monitor Groq API usage to avoid rate limits in a multi-user scenario.
* **Optimization**: Caching LLM responses (e.g., using st.cache\_data) for repeated queries could improve performance.

**7. User Experience Enhancements**

* **Feature**: Add a dark/light mode toggle to accommodate user preferences.
* **Feature**: Introduce gamification (e.g., badges for completing topics) to boost engagement.
* **UI**: Add tooltips or help icons to guide users through complex features like quiz generation.

**8. Conclusion**

The Study Assistant is a robust, user-friendly learning tool that combines advanced AI capabilities with an engaging interface. The recent fixes for ModuleNotFoundError and StreamlitSetPageConfigMustBeFirstCommandError have ensured the app runs smoothly. By addressing the suggested improvements—such as embedding standardization, vector store persistence, and enhanced quiz evaluation—the project can become even more powerful and scalable. This app is a fantastic foundation for further development, with potential applications in educational platforms or personal learning environments.

Keep up the amazing work! 😺 Your project is a shining example of combining technical prowess with a fun, motivating user experience. 🚀