Tel Hai Study Buddy

In this project, you are going to research, design and implement a chatbot that bases its answers on multiple types of media sources. Your client is Tel Hai college, and this product is intended to be used by all of the students, regardless of the degree they are studying. During the given time period, your goal is not to build an entire product (not enough time), but to prove that building this kind of product is possible by implementing a quick and dirty MVP of it, and deploy it for the students to interact with.

Product description

Core features

Multimodal Search:

- Allows users to search for lecture materials across multiple formats: documents, video recordings, and audio files.
- Provides results that are timestamped for video and audio, guiding users directly to the relevant parts of the **original input.**

Natural Language Query Support:

- Enables students to ask questions or search for content using conversational language, such as requesting specific topics, dates, or lecture details.
- Recognizes and responds to both specific and broad queries about lecture material.

For the core product, GUI is not required.

Optional features

After you're done with the core features, choose any subset of features you'd like from this list and implement them. Start with the features that you think are the most cost effective (low cost, high value for the user), **the features are not numbered by priority**. In any one of these features, before implementing the actual product, create a basic POC (proof of concept) and call the staff for review.

1. **GUI**
    - Create a basic user interface (web\app) for your product. Pay attention - in this feature in particular, make sure to search for existing tools and use them. When done with the research, call the existing staff.
2. **Personalized Recommendations**:
    - Suggests lecture materials, readings, or supplementary resources based on previous searches and user preferences.

3. **Real-Time Assistance**:
    - Interacts with users in real-time to answer follow-up questions, clarify concepts, and provide additional related material (Requires some kind of GUI/CLI).
4. **Advances filtering features**:
    - Allow the users the option to choose, during chat setup, which types of content (Video, Audio, Documents) they want the answers to be based on, allowing them to choose only a specific content type.
5. **Support multiple users:**
    - Support multiple users in your product.
    - Research authentication ways, show the staff the results of your research before starting to implement.
6. **Manual Content Bookmarking**:
    - Enables users to bookmark or save specific Audio\Video\Documents, with a specific timestamp for future reference and quick access.
7. **Automatic Content Categorization**:
    - Automatically tag your materials based on its content, when first extracting the data from them.
    - Allow users to search for specific tags.
8. **Voice Interaction Support**:
    - Allows users to perform searches and interact with the chatbot via voice commands for a hands-free experience.
9. **Recognize the lector:**
    - Think of a way to distinguish between students and lectors, and include that in the extracted text.
10. **Auto-Completion & Query Suggestions**:
    - Suggests potential queries and auto-completes as users type, helping them refine their searches and find content more efficiently. Think about a way to integrate your Google project with this one.
11. **Multi-Language Support**:
    - Supports multiple languages to cater to diverse academic needs and to ensure accessibility for all students. Which languages are important to support? Document your reasoning
12. **Multiple input formats:**
    - Support additional input formats (file types\extensions) of video, audio or text. Which formats are important to support? Find out, and document your reasoning

Research

In this exercise, you should use external tools and libraries as much as you can, in order to reduce the amount of code you have to develop and maintain yourself.
Hugging face is a good place to start your search. A research phase is a crucial part of a development cycle, when your product contains new concepts to the team. Its length varies according to the project, and the amount of concepts you have to develop from scratch.

Input parsing

Your goal for the input parsing phase is to convert all the different media types to text.
There are a lot of technologies available that could do that, most of them are AI based.

Step 1: Audio Transcription

Goal: Research technologies that convert audio into text and create a PoC in a Jupyter notebook.

- **Research Focus**: Identify multiple tools or algorithms for converting spoken language into text, considering factors like noise, language support, and performance.
- **Testing Requirement**: Each team must test at least two different transcription technologies on a variety of audio samples (e.g., different accents, background noise, varying audio quality). Analyze accuracy, speed, and handling of edge cases.
- **PoC Task**: Demonstrate the transcription of multiple audio files using the technologies researched. Compare the results (text accuracy, processing time) in your notebook and discuss strengths and weaknesses.In addition, you can choose to implement any subset of features you'd like from this list. Choose to start with the features that you think are the most cost effective (low cost, high value for the user)

Step 2: Image Recognition in Video & Audio Transcription

Goal: Research technologies for recognizing objects or scenes in video, and create a PoC that transcribes video audio.

- **Research Focus**: Explore at least two technologies for object detection in video frames and audio transcription from video.
- **Testing Requirement**: Run multiple tests on different video samples, including videos with varying resolutions, lighting conditions, and object complexities. Transcribe audio from these videos using the selected tools.
- **PoC Task**: In the Jupyter notebook, showcase the processing of several video clips, highlighting object detection and audio transcription accuracy. Include performance metrics such as processing speed, object detection accuracy, and quality of audio transcription. Compare how different tools perform on each task.

Step 3: Document Parsing

Goal: Research document parsing technologies and implement a PoC in Jupyter to extract structured data from documents (e.g., PDFs or scanned images).

- **Research Focus**: Investigate multiple methods for parsing documents, focusing on the ability to handle structured and unstructured formats, as well as noisy or low-quality scans.
- **Testing Requirement**: Test at least two parsing technologies across various document types, including clean PDFs, scanned documents with noise, and complex forms with tables. Evaluate extraction accuracy and speech

Retrieval-Augmented Generation (RAG)

Your goal in the RAG phase is to implement a system to retrieve relevant data for a given query.

Step 1: Text Embeddings

Goal: Explore technologies that transform text into embeddings, creating a PoC to evaluate their performance on a diverse range of textual inputs.

- **Research Focus**: Identify several models that convert text into vector embeddings, considering factors such as language coverage, embedding dimensionality, and how well the embeddings capture contextual information.
- **Testing Requirement**: Experiment with at least two different embedding models on various text types (short vs long text, technical vs conversational language). Assess each model's ability to generate meaningful embeddings in terms of semantic similarity and computational efficiency.

Step 2: Vector Database

Goal: Research and evaluate tools for storing and querying embeddings, producing a PoC that enables scalable and efficient embedding searches.

- **Research Focus**: Analyze different vector databases for their performance, scalability, and ease of integration. Consider key factors such as real-time search capability, handling large datasets, and indexing speed.
- **Testing Requirement**: Test at least two vector databases, benchmarking them for query performance on both small and large datasets. Evaluate search precision, time to index new embeddings, and scalability under high query volumes.

Step 3: Generation

Goal: Explore how large language models can generate content based on embeddings retrieved from a vector database, producing a PoC to evaluate their generation quality.

- **Research Focus**: Investigate different models that leverage content from vector databases to produce context-rich, relevant responses. Assess the models based on their fluency, coherence, and ability to integrate retrieved content meaningfully into the generated output.
- **Testing Requirement**: Test two or more generation models by using them to answer questions based on retrieved text or documents. Measure their output for relevance, completeness, and contextual accuracy, particularly in challenging or ambiguous scenarios.

MVP Implementation

Step 1 - System Design

Goal: After researching and selecting the appropriate tools for your product, it's time to design your system. Focus on creating a modular design that follows the principle of separation of concerns.

Workflow Key Points:

- **MVP Focus**: You are building a Minimum Viable Product (MVP), so keep the design process efficient—*limit design discussions to one hour*.
- **Apply Course Concepts**: Use the principles and techniques learned during the Python course, particularly those from the *final exercise*.
- **Diagram**: Choose a free diagramming tool to visually represent your system architecture (draw.io or similar tool).
- **Team Involvement**: Not all team members need to be involved in the initial design process. Assign a smaller group for planning, and once the design is complete, conduct a **design review** **with the entire team** to gather feedback and ensure everyone is aligned. ***Call the course staff for the design review as well.***
- **Clear definition:** for each component, define its function in a single sentence.

Design Key Points:

1. **Database Interaction**:
   - Identify which parts of your program will need to interact with the database.
   - Decide what information needs to be stored (e.g., parsed data, metadata, user interactions).
   - Determine the type(s) of database required (e.g., SQL, NoSQL) based on the nature of the data.
2. **Component Communication**:
   - Define how the different modules of your program will communicate with one another. Make sure that your communication supports large files as well.

    - Ensure clear separation between input parsing, database operations, and chatbot logic.
3. **Parsed Input Structure**:
    - Design a **unified or near-unified data structure** that can represent the parsed results from all input types (audio transcription, video image recognition, document parsing).
    - This structure should be consistent and flexible, allowing for easy storage and retrieval of the processed data.
4. **Reference to Original Files**:
    - Plan how the chatbot will base its responses on the original files, not just the parsed data.
    - Consider linking the parsed data back to the original files, using references, timestamps, or other contextual metadata.
5. **Scalability**:
    - Ensure the system design can handle increasing amounts of data or users.
    - Think about how the database, processing pipeline, and communication between modules will scale as the system grows.
6. **Outwards API:**
    - The outward-facing API serves as the primary interface for external clients to interact with your system. It must be designed to efficiently provide access to the system's core functionalities, such as submitting files for processing, retrieving results, and querying the chatbot. Clients should be able to seamlessly send data and receive responses, with clear endpoints that guide their interactions.

Step 2 - Task Breakdown and Implementation Planning

Goal: Once the system design is approved by the team, it's time to break down the design into smaller coding tasks and focus on defining clear interfaces between program components. This phase ensures the tasks are manageable, each part of the program is well-defined, and all pieces work together cohesively.

Workflow Key Points:

- **Task Granularity**: Break down the system design into small, manageable tasks that can each be completed within a short timeframe (e.g., 2-3 hours each). Ensure each task has a **Definition of Done (DoD)**, which clearly defines what it means for the task to be considered complete.
- **Interface Definition**: Define the interfaces and interaction points between different parts of the system to ensure smooth integration.
    - **Interaction Points**: For each module or component, clearly define the interfaces. Consider:
        - What input data is expected (e.g., parsed text, video frames).
        - What output data should be provided (e.g., formatted responses, database entries).

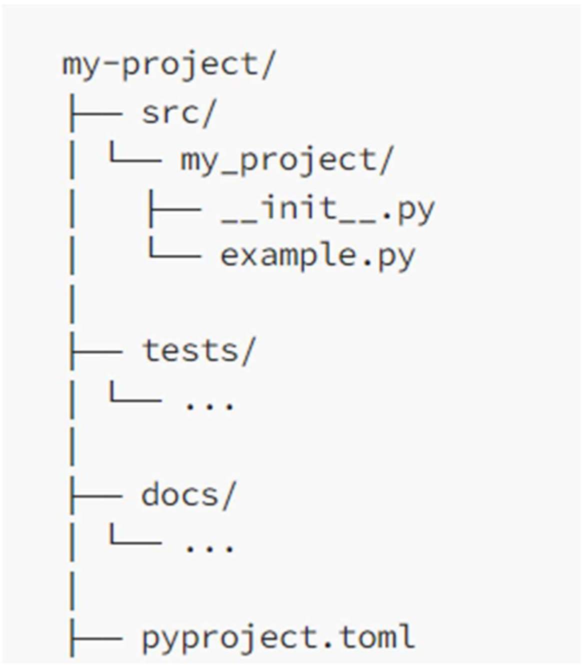
        - How the data will be passed between modules (e.g., API, function calls, etc.).
        -
    - **Data Formats**: Ensure consistent data formats across modules, especially in terms of the unified data structure. All components should be compatible with the common structure you designed earlier.

- **Focus on Collaboration**: Ensure that tasks are broken down in a way that allows for parallel development. This minimizes dependencies between team members and allows multiple parts of the system to be developed simultaneously.

Step 3 - Implementation

Goal: A working, basic mvp of your product. Your project should

- Use git -
    - Master should contain a working version of your project at all times.
    - Use pull requests
    - Use small, and meaningful commits with indicative commit messages.
- Be tested using at least a single, happy flow, system test, for each input type.
- Use Python venv
    - How to create python venv
- Be in the following structure, and use .toml file.



```
my-project/
├── src/
│  └── my_project/
│     ├── __init__.py
│     └── example.py
│
├── tests/
│  └── ...
│
├── docs/
│  └── ...
│
├── pyproject.toml
```

Each piece of code should comply with the following standards -

- A single test that should cover the "happy path" or expected workflow of the function (i.e., testing the function with valid inputs).
- Logging - use indicative log levels.
- Use type hints and documentation when needed.

Step 3 - Containerization

Goal: Containerize your MVP with docker to ensure the system is portable, easily deployable, and consistent across various environments. This step allows different components of your system (input parsing, database, chatbot logic, etc.) to run in isolated, standardized environments.

Workflow Key Points:

- Focus on MVP: Only containerize the core modules needed for the MVP, such as the input parsing service, the database, and the chatbot logic.
- Keep Efficiency: Limit the containerization process to essential services and keep configuration simple for the MVP stage.

Containerization Key Points:

1. Identify Core Modules to Containerize:
    - Input Parsing Module: Audio transcription, video image recognition, and document parsing services.
    - Database Module: each database should run in its own container.
    - Chatbot Module: Include the retrieval-augmented generation (RAG) system that processes the parsed data and user queries.
2. Create a Dockerfile for Each Component:
    - For each core module (input parsing, database, chatbot logic), create a separate `Dockerfile`.
    - Input Parsing Service Dockerfile: Specify the dependencies for running tools like Whisper, Vosk, BLIP, Tesseract, etc.
    - Database Dockerfile: For your database container, ensure that the appropriate database is installed, and persistent volumes are mounted for data storage.
    - Chatbot Service Dockerfile: Include the dependencies for running your chatbot logic, including the Python libraries for RAG, text embeddings, and communication with the database.
3. Set Up Docker Compose:

    - Use Docker Compose to manage multi-container applications. This ensures that each service (input parsing, database, chatbot) runs together and communicates properly.
    - In your `docker-compose.yml`:
        - Define services for each core module.
        - Expose the necessary ports for communication between containers.
        - Link services (e.g., chatbot service connected to the database service).
        - Include volume mounts for data persistence and storage of parsed input files.
4. Environment Variables:
    - Define environment variables for each service in a `.env` file. This could include:
        - Database connection strings.
        - Paths for storing and accessing files.
        - API keys for external services.
        - Model paths for text embeddings or pre-trained models.
        - Flag for development or production environment.
5. Testing and Debugging:
    - Test each container in isolation to ensure it's working properly.
    - Use Docker's built-in logging (`docker logs`) to track and debug issues across different containers.
    - Once individual containers work, test the entire system with Docker Compose to ensure communication and data flow are seamless.

Deployment

Goal: deploy your containerized system on a scalable cloud infrastructure, enabling robust and flexible handling of user queries.

Deployment Key Points:

- Upload your Docker images to GCP artifact registry
- Explore GCP's infrastructure services for deploying containerized applications, focusing Cloud Run for deployment of your product.

Logging and Monitoring:

- Use **Google Cloud Monitoring** to track real-time performance metrics (e.g., latency, CPU usage, memory consumption).

- Set up automated alerts for key metrics (e.g., high memory usage, slow database queries) so the team can react quickly to performance degradations.
- Store logs using **Google Cloud Logging** for auditing, troubleshooting, and post-mortem analysis.

The factors you should measure are -

- Results quality
    - Read about methodologies for comparison between different models, and pick an cost effective method
- Efficiency (performance profiling)
- Memory usage (memory profiling)

When done with any of the steps, call the course staff for review