Bindr

Eric Cheng, Aasiyah Shaikh, Ibrahim Yurdan

Video

Introduction

Bindr is a web based application that acts as an organizational tool for students. It does this by providing students with an interactive, personalized tool to organize study material, practice concepts, and keep track of deadlines. It has a few different features, including a document chat, file hub, calendar, and a study plan creator. It utilizes React, Nodejs, Python, Firebase, and OpenAI api.

Installation Steps

1. Install Nodejs
2. Download and unzip the project
3. Using anaconda prompt, find the project directory and type `conda env create -f environment.yml`
    a. Then activate the environment using `conda activate bindr`
    b. If on windows, type `set OPENAI_API_KEY=<insert api key>` or if using unix, type `export OPENAI_API_KEY=<insert api key>`
    c. Go to the backend directory and run app.py `python app.py`
4. In another terminal, go to the project directory and type `npm install` to download dependencies
5. Then type `npm start` and go to http://localhost:3000/ in your browser

Document Chat

The Document Chat feature allows users to upload documents (e.g. PDFs) and interact with their content by asking questions. It utilizes the OpenAI API to extract meaningful responses from the uploaded document. Once a document is uploaded, the system processes the text using PyPDF2 and stores the extracted content in Firebase Firestore. Users can then ask

questions related to the document, and the application combines the document content with the user's query to provide accurate answers.

Challenges:

- Properly extracting text from complex PDFs was a challenge, especially those with scanned images or unusual formatting.
- API rate limits and ensuring the queries were contextually relevant required fine-tuning the prompts.
- Prompt Engineering: We tried to get the assistant to answer based only on provided documents, for example a note sheet with very specific information. However, the API took this extremely literally and would only directly quote text. It was a challenge to come up with a prompt that allowed the API to answer based on provided information only but still be more intelligent than a basic search tool. This is a prompt that did not work well:

```
prompt = (
    "When user greets you, respond with a greeting and explain your role. "
    "Answer the question based on the provided document content. "
    "If the answer cannot be determined from the document, respond with 'Hm...I can't seem to find the answer to that in your document. '\n\n"
    f"Document Content:\n{file_content}\n\n"
    f"Question: {question}"
)
```

This prompt worked much better:

```
prompt = (
    "If the user greets you, respond with a greeting and explain your role only once. "
    "Analyze the file content and answer the question based on what you learn. "
    "If the answer cannot be determined from the document, respond with 'Hm...I can't seem to find the answer to that in your document. '\n\n"
    f"Document Content:\n{file_content}\n\n"
    f"Question: {question}"
)
```

Highlights:
The feature successfully integrates document storage, text extraction, and AI-powered responses in a seamless interface.

My Documents

The My Documents feature acts as a file hub where users can view, manage, and select previously uploaded documents for interaction. It lists all uploaded documents stored in Firebase Storage and retrieves associated metadata from Firestore, such as URLs and extracted text.

Challenges:

- Implementing efficient database queries to fetch document lists without delays.
- Ensuring data synchronization between Firebase Storage and Firestore.

Highlights:
A simple and user-friendly interface for document selection and management was achieved.
The integration between storage and database worked smoothly after initial debugging.

Calendar

The Calendar feature in Bindr allows users to seamlessly integrate their syllabus or documents containing important deadlines into their study plan. When a user uploads a syllabus (typically in PDF format), the system extracts key dates and events. These dates are then automatically added to the calendar, providing students with a clear visualization of their upcoming tasks and deadlines.

This feature is particularly helpful for students who want to stay on top of their assignments, exams, and other academic commitments without manually inputting each date. By linking the extracted dates to the calendar, students can plan their study schedule more effectively and never miss a deadline.

What Worked:

- The integration of PDF parsing and regex-based date extraction proved effective for identifying standard date formats (e.g., MM/DD/YYYY or "January 5, 2024").
- Automating the addition of extracted dates to the Firebase calendar made the process seamless and user-friendly.

What Didn't Work:

- Some syllabus files with unconventional formatting or images of text posed challenges for date extraction.
- Occasionally, extracted dates lacked associated context, such as event descriptions, which required additional manual input by the user.

Study Plan

The study planner uses user-inputted parameters to generate a study plan. This includes a timeline, weekly availability, choose syllabus, specific topics, and study preferences. We used prompt engineering to instruct the assistant how to use the information. If the user indicated that they were free the whole day, the assistant was instructed not to fill up the entire time with studying, but instead use time estimates for how long it takes to cover specific topics.

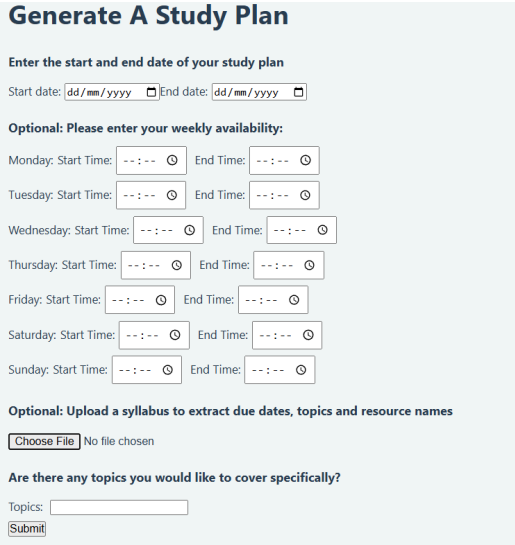
Frontend

The user interface keeps a consistent blue and white theme throughout the entire application, which is bounded by the navigation bar. It employs many different techniques and tools such as containerization and react elements to make the application more appealing than base html.

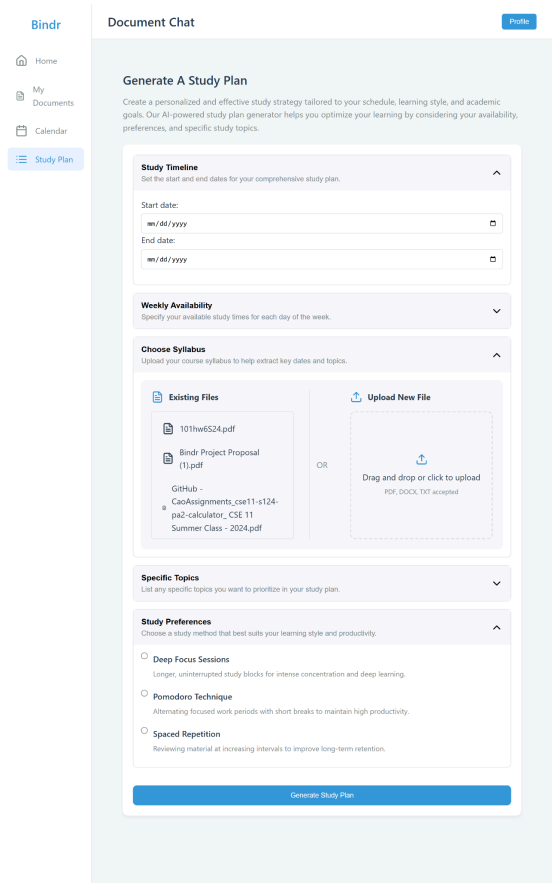
Although I do have experience with regular html, css and javascript, I had never used frameworks like react and node and learning them took a bit of effort. However, it did make a lot of things easier, such as including icons and plugins like the calendar. An interesting problem I

found was rendering, which was especially a problem when formatting the study plan. That after the user made some changes, it would essentially rerender the entire page, either putting the user-input out of focus or worse, not acknowledging that anything was inputted. This was fixed by adding javascript functions to handle each individual change. There's still a couple of UI problems that I couldn't find a fix for, such as the navigation bar changing sizes when certain elements like the calendar take up a sizable portion of the body.

Before:



After:



Sources

For the study plan, we wrote most of the code ourselves, but asked ChatGPT to add some functionality and debug. For example, we asked ChatGPT to help create the form data in the right format to send to the backend function. In addition, the more complicated formatting was done with AI, i.e. the weekly availability/day selector. ChatGPT also helped us set up the OpenAI API, as we were having trouble with it. AI was occasionally called to help with processing PDF files, extracting text, and finding dates in the files, as it involved packages we were unfamiliar with.