📝🚀 Coursepilot

6.0 Software Design Description

6.1 Introduction:

With innovative tools and frameworks emerging surrounding LLMs we wanted to apply some new techniques such as RAG (Retrieval Augmented Generation) to provide value to students by making learning more effective and engaging. Our idea is to embed the semantic meaning of documents, which could be student-taken notes or slides uploaded by a professor, store the documents in a vector database, and use them to provide greater context to generate a better output when the LLM is used. Coursepilot's backend will be the RAG engine, that will parse documents, embed their semantic meaning, and store each document in a MongoDB Atlas database with an id, the text, and its vector embedding. This vector embedding allows us to retrieve relevant documents when querying later. We will also be making our Gemini API calls from the backend and connecting it to the frontend with Flask. The frontend will be mainly built with Next.js, and will handle user authentication with Clerk.

6.1.1 System Objectives

The objective of this application is to create a user interface to accelerate note taking using AI, providing study tools, quizzes, and a jumpstart for presentations and projects. The goal of this application is to help teach its users, without causing them to avoid learning.

6.1.2 Hardware, Software, and Human Interfaces

6.1.2.1 Hardware Resources

Resource	Dev	Execution
Windows PC	X	X
AMD Ryzen 5 5600 3.8GHz 6 core	X	
32gb DDR5 RAM, Nvidia 3070 8gb VRAM	X	
Macbook Pro	X	X
Apple M1 Max	X	
32gb RAM	X	

6.1.2.2 Software & Human Interface Resources

Resource	Dev	Execution
Next.js	X	X
Flask	X	X
MongoDB	X	X
Heroku		X
Visual Studio Code	X	
Cursor	X	
Clerk Auth	X	X
Google Gemini LLM	X	X
TipTap	X	X

6.2 Architectural Design

6.2.1 Major Software Components

- 6.2.1.1 Classlist - a list view for organization displaying all classes added by the user
    - Emoji picker - an emoji picker to select icons for different classes
    - Title - a customizable title input for class name
    - Backend - selection updates note in storage
- 6.2.1.2 Notelist - a list view for organization displaying all notes within the selected class
    - Emoji picker - an emoji picker to select icons for different notes
    - Title - a customizable title input for the note
    - Backend - selection updates note in storage
- 6.2.1.3 Text Editor - a rich text editor for the selected note
    - Menubar - a menu of buttons to control text options
        - Connection to the text area and its selections
        - Bold - button to bold text
        - Italics - button to italicize text
        - Lists - button to create a numbered/bulleted list
        - Size - button to increase/decrease font size
        - Color - a picker for font color
    - Editor - the main text area
        - Connection to the menu bar and its selections
        - Formatting - markdown support for headings, lists, code blocks
        - AI - a debounced autocomplete
            - Preview - a gray text preview of autocompleted text that can be filled using tab
            - API - request made to Gemini using the context of the user's notes
        - Backend - modification updates note in storage
- 6.2.1.4 Quizzes - a list view for generated flashcards from notes
    - Flashcard - a front/back Q&A
        - Front - a question to be answered by the back of the card
        - Back - the answer to the question on the front of the card
- 6.2.1.5 Document chunker - currently implemented as a sentence based chunker but can be extended to perform semantic chunking
- 6.2.1.6 Document embedder - utilizing the BERT (Bidirectional Encoder Representation for Transformers) language model to capture semantic meaning of text chunks in vector representations

- 6.2.1.47 MongoDB Atlas Vector database - allows for storage of text data along with its vector representation, which allows for effective querying and retrieval using the KNN algorithm
- 6.2.1.8 Flask server hosting an API will create a data pipeline from the RAG engine to the frontend
    - “/autocomplete”
        - Takes the context of the current note and returns the suggested autocomplete
    - “/getContext”
        - (optional) Takes the current note and provides metadata for tooltips for related documents previously uploaded

6.2.2 Major Software Interactions