**SYLLABUS CHECKER**

****

**TEAM**

Jack Demtshuk, Ishaq Halimi, Joscelin Montoya-Rojas, Rebecca Nanayakkara, Nicholas Taweel

**CLIENT**

Dr. Judith Ozment

Division Head of Science and Engineering

o96@psu.edu

Problem Statement

A syllabus is one of the most important documents a student receives, yet its quality and completeness can vary widely from course to course. Missing grading policies, unclear attendance expectations, or outdated accessibility statements can leave students confused and unprepared. Faculty at Penn State Abington are required to follow the detailed guidelines outlined in the Faculty Handbook, but checking every syllabus against these requirements can be time consuming and inconsistent. Current tools like Grammarly may help with grammar, but they cannot determine whether a syllabus contains all the elements that students and administrators expect.

The Syllabus Checker aims to solve this problem by providing an automated system that evaluates syllabi for both compliance and clarity. Our tool will go beyond grammar checks by identifying missing or outdated content, assessing readability, and generating actionable recommendations for improvement. In doing so, it supports instructors in creating documents that not only meet institutional policies but are also student friendly and accessible.

By automating this process, the project offers benefits across the board. Students will receive clearer, more reliable syllabi that help them plan and succeed in their classes. Faculty will save time and reduce the stress of manual reviews, while administrators can be confident that institutional standards are consistently met. In short, the Syllabus Checker will raise the overall quality of syllabi and improve the academic experience for everyone involved.

Related Work

When it comes to checking documents such as syllabi, there are already some tools out there that can handle parts of the job, but none of them really cover everything we are aiming for. For example, while Grammarly is great at catching grammar mistakes and spelling errors, it doesn’t know what specific content a syllabus is supposed to have. Grammarly won’t flag your syllabus if something important is missing, such as the grading policy, office hours or accessibility statements.

At our university, faculty have access to Appendix E of the Faculty Handbook, which lays out very detailed syllabus guidelines. These policies specify exactly what must be included, such as instructor contact information, office hours, attendance expectations, and more. In addition, according to the Faculty Handbook, copies of all syllabi have to be submitted to the professor’s division head within the first ten days of the semester. This means that reviewing these syllabi can be time consuming and requires a lot of responsibility for each syllabi to be manually checked for the required details.

This is where our project is different. The Syllabus Checker combines the strengths of existing tools while also focusing specifically on what makes a syllabus complete and student friendly. Our tool will not only fix grammar, it will look for required information, flag outdated content, and give recommendations on how to make the syllabus more readable. Ultimately, our tool is designed to support both professors and students, while also saving time for administrators who are responsible for reviewing syllabi.

Features

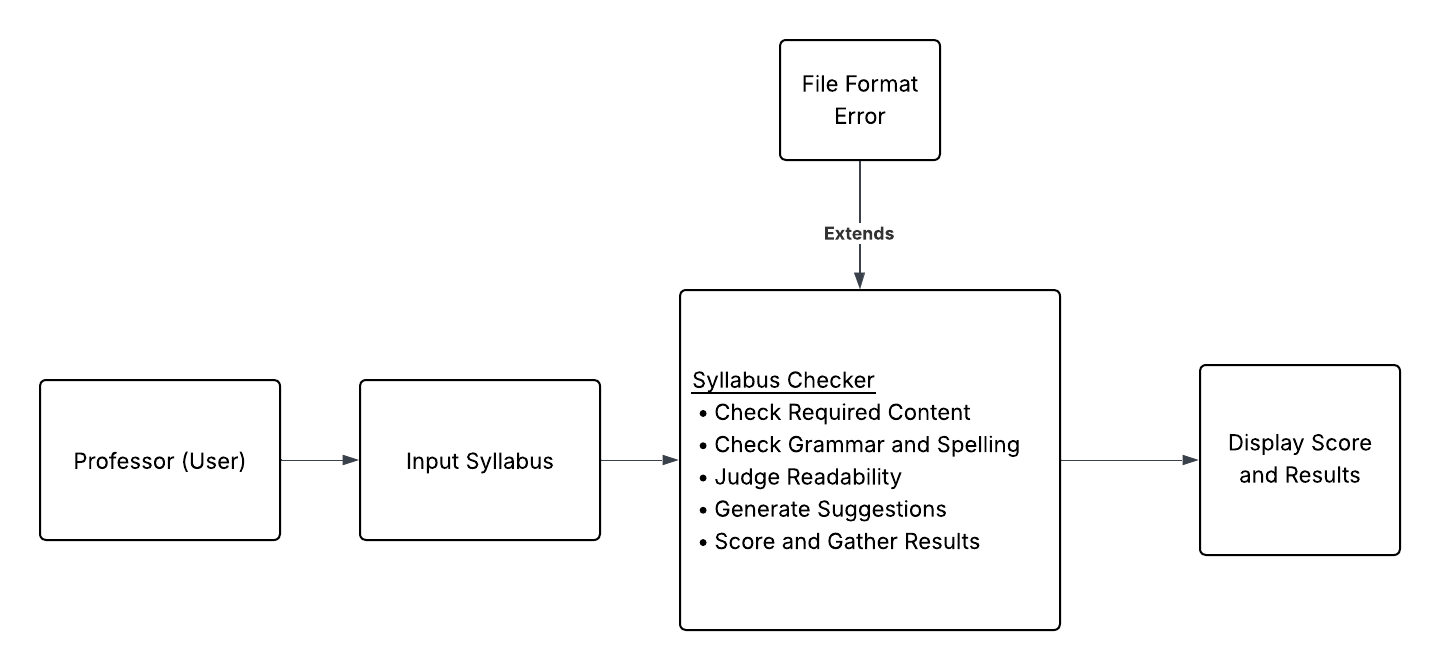
The “Syllabus Checker” tool we wish to create will scan syllabi for their quality and accuracy. The tool will check for certain required information from a user’s input syllabus, as well as see that it is grammatically correct and comprehensible. The tool would then output a list of sections that it found to be incorrect, a list of sections that it found to be insufficient, among other descriptive flags. Much of this will be detectable by a machine learning model that we hope to create alongside this tool. Things that are not detectable through the machine learning model will be hard coded using binary trees and conditional statements. Syllabi must be in a PDF format, so our tool can scan text directly from the file. If a syllabus were uploaded as an image file, like a .jpg or .png, it would require some form of computer vision and image processing.

Distinct information that the tool will check for include professor’s name, a grading policy, a course description, and required resources and hotlines. Some sections, such as the grading policy, will not be judged on its perceived fairness or quality as that is dependent on the professor and teaching style. Some courses or departments may have their own requirements. A case of this would be a lab course needing to outline lab safety. The naming and structure of each section will not require specific keywords. For example, a section titled “Grading Policy” is as valid as if it were named “How I Grade.” The program will rank syllabi on whether they pass or fail the check, by providing a grading scale where certain aspects that are required must be present or else they automatically fail, and aspects that are highly recommended are not present, or content is insufficient or overly complex, it will pass but with points deducted. When judging these sections, it will check that there is sufficient information. If a particular section is less detailed than it typically is, the tool will flag it. It will also check for the opposite, flagging where

parts are less simple than they should be. It will provide recommendations to the user on how to improve the syllabus based on the sections that fail to reach a passing grade. The tool will also check for syntax and grammar errors.

The tool is meant to be easy to use, with the expected users just being professors. Therefore, the interface will be very simple and only reliant on the inputted syllabus. We hope to make the output clear and concise, highlighting where the syllabus is lacking and recommending changes.

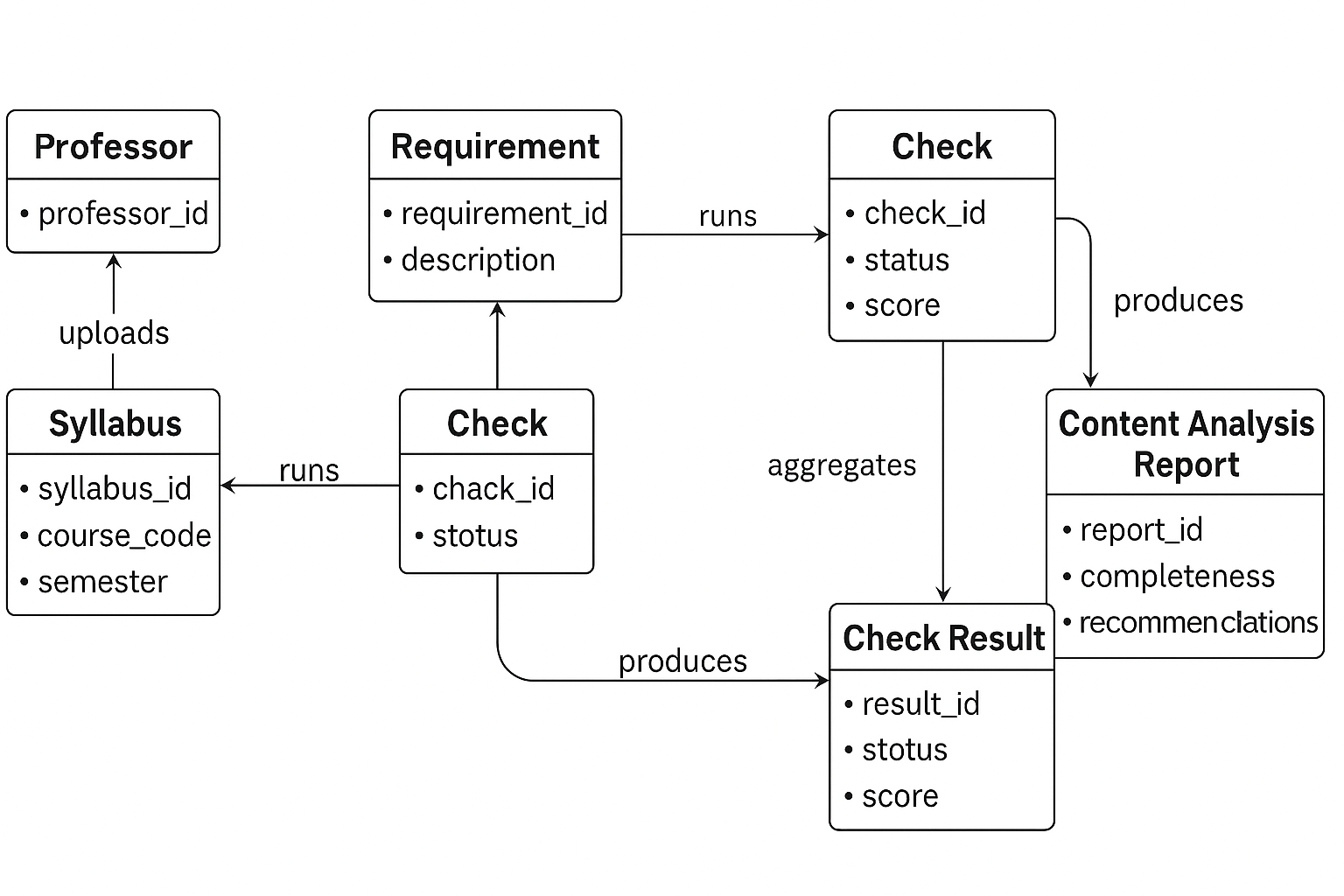
Use Case Diagram

****

**Figure 1. Use Case Diagram**

The use-case situation involves the professor inputting a syllabus into the syllabus checker, where the program then judges the content and quality of the syllabus. A file format error may be raised, which is an extension of the syllabus check step. Given that no error occurs, the syllabus checker will display a score and suggestions.

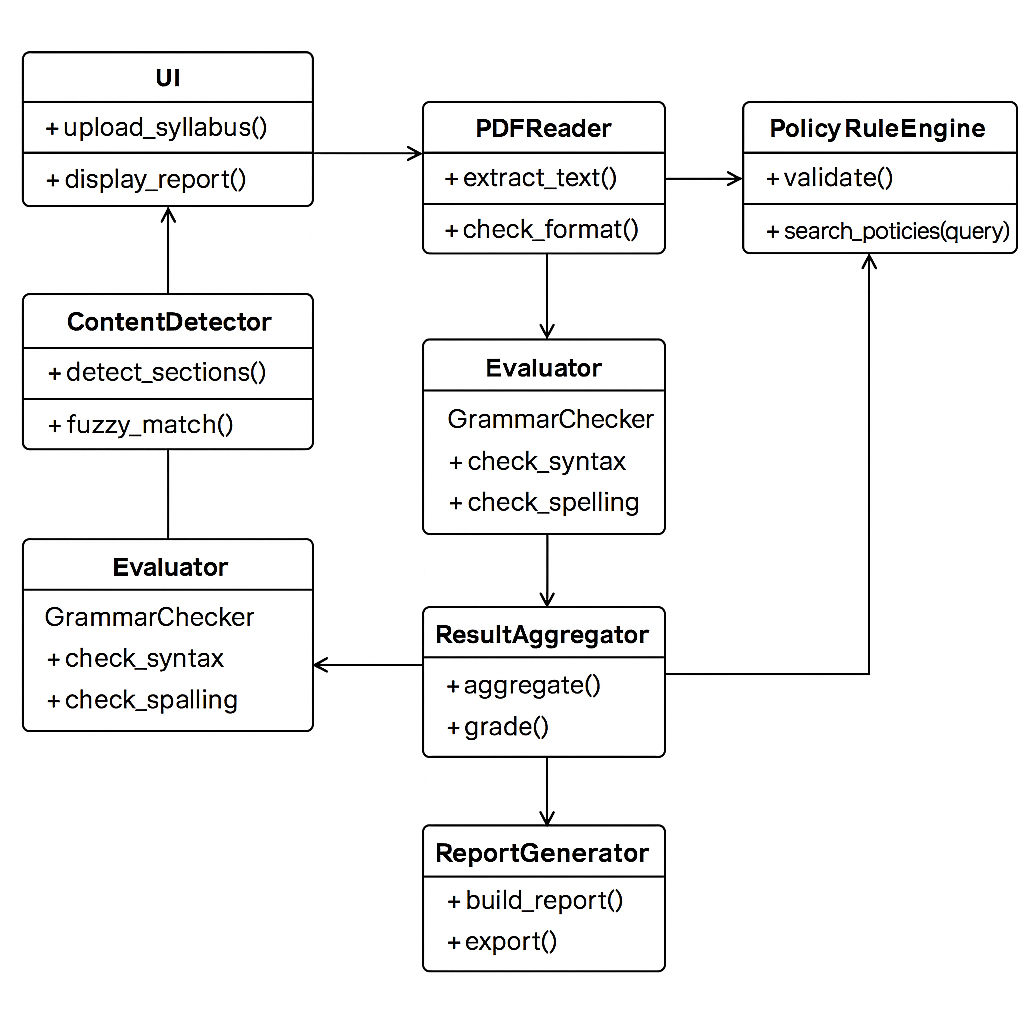
Design Specification



**Figure 2. ER Diagram - Core Data Relationships in the Syllabus Checker**

The ER diagram above illustrates how data flows through the Syllabus Checker system. A Professor uploads a syllabus, which is associated with a specific course. Each syllabus is evaluated against a set of requirements defined by the university’s Faculty Handbook and Appendix E. These requirements generate corresponding checks that analyze the syllabus for completeness and accuracy.

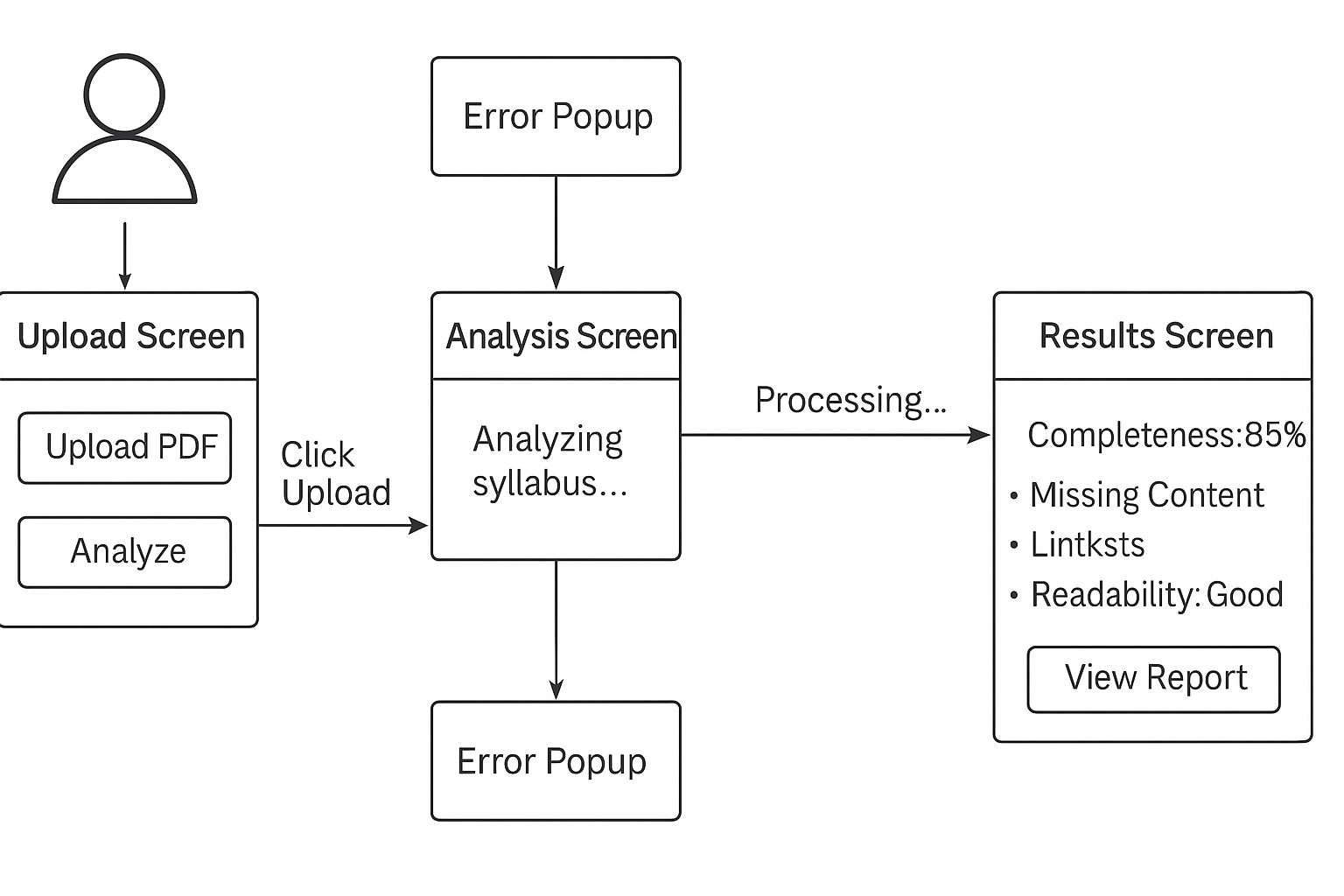
Each Check produces a Check Result, which captures details about whether the required information was found, missing, or outdated. These individual results are then accumulated into a content analysis report, which summarizes the overall syllabus quality, readability, and compliance with institutional guidelines. The requirement entity also informs the final report by providing the context for each evaluation criterion. This structure ensures that every syllabus is assessed systematically and that feedback is accurate, transparent, and consistent across all courses.

****

**Figure 3. UML Class Diagram — Core Program Architecture of the Syllabus Checker**

The UML Class Diagram illustrates the structural design of the Syllabus Checker system and how its classes interact. The UI class enables professors to upload syllabi and view results. The PDFReader class extracts text and validates the file format, while the ContentDetector identifies required sections through fuzzy matching. The PolicyRuleEngine validates content against institutional policies, and the Evaluator (which includes the GrammarChecker) reviews syntax and spelling for clarity.

Outputs from these modules are compiled by the ResultAggregator, which calculates a completeness score and summary data. The ReportGenerator then creates a final content analysis report displayed through the user interface. This modular design supports easy maintenance, scalability, and clear organization of system responsibilities.



**Figure 4. User Interface Diagram — Screen Flow of the Syllabus Checker**

The User Interface Diagram illustrates the user interaction flow for the Syllabus Checker system. The process begins with the upload screen, where the professor selects and uploads a syllabus file. If the uploaded file is not in the correct format (for example, not a PDF), an error popup appears to notify the user. Once a valid file is uploaded, the system transitions to the analysis screen, which displays the progress of the syllabus evaluation, including content verification, grammar and spelling checks, and readability assessment.

After the analysis is complete, the program navigates to the results screen, which presents a comprehensive report including the syllabus completeness score, missing sections, readability score, and actionable recommendations for improvement. The layout is designed with simplicity and clarity in mind, ensuring that professors can easily interpret results and make necessary adjustments to their syllabi.

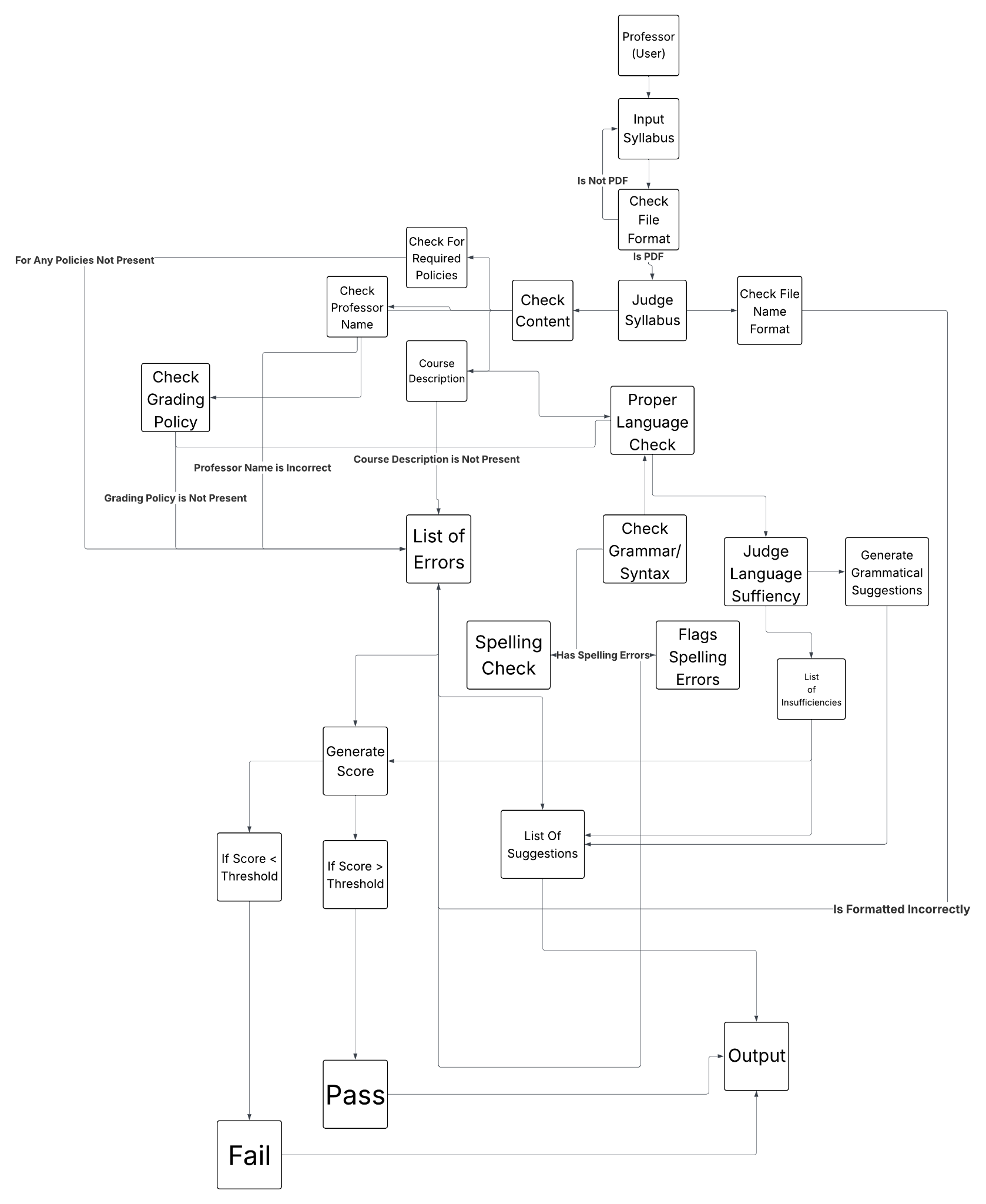
Required Tools and Technologies:

Our project uses a range of tools and technologies that allow us to read, analyze, and evaluate syllabi efficiently. The program is built in Python 3.11, which provides the flexibility and libraries needed for text analysis, PDF handling, and machine learning integration. To read syllabi, we use PyPDF to extract text directly from uploaded PDF files, preparing them for analysis. Once the text is processed, the program analyzes it using both the RapidFuzz and Sentence Transformers libraries. RapidFuzz performs fuzzy keyword matching to find important sections even when worded differently, while the CrossEncoder model from Sentence Transformers applies machine learning to understand the semantic meaning of text. This allows the program to detect required content based on context and phrasing, rather than relying solely on exact keyword matches.

Grammar and spelling are checked using TextBlob, which helps improve clarity and professionalism across documents. The graphical user interface is built using Streamlit, allowing professors to upload syllabi, run the analysis, and view the results in a clear and user-friendly layout. Streamlit connects directly with our backend analyzer and spell checker, which together evaluate whether a syllabus meets the required standards and passes the overall quality check. Additional tools such as NumPy and regular expressions (re) are used throughout the project for sentence processing, text handling, and similarity scoring.

In the future, we plan to add a few practical tools to make the Syllabus Checker more complete and efficient. One improvement would be adding a small database, such as SQLite, to store past syllabus reports so professors and administrators can review results without re-uploading files. We also plan to expand the Streamlit interface with additional pages that display graphs or summaries of analysis results, giving users a clearer overview of where syllabi commonly fall short. Another possible addition is refining our machine learning model to detect more complex context, such as distinguishing between missing content and incomplete explanations. These updates would strengthen the system’s usability and accuracy without requiring a major redesign or heavy new infrastructure.

UML Class Diagram

****

**Figure 5. UML Class Diagram**

The program prompts the user to input a syllabus as a PDF file. It will first check to ensure that the file format is correct. If it is not, it will reject the input and ask for a new one. If it is, then the program will continue. The program then diverges into performing many analysis tasks on the input, such as required policies, name, spelling and grammar, and general readability. The program then produces a list of suggestions based on the results of these analyses, as well as generates a score to rate the input. The program will then output the input’s score, and whether they pass or fail the check. Some aspects of the input may cause automatic failure, such as missing required content. After receiving the score on the output, it will also provide the list of suggestions on how to improve the syllabus, regardless of score or pass/fail.

Work Plan

1. **Week of September 23: Acquire PSU Guidelines and Recommendations**

We will start by reviewing all syllabus policies and guidelines from the Penn State Abington Faculty Handbook (2025-2026) to understand what is required in a syllabus, as well as look into and review what is recommended to be included in a syllabus by Penn State.

1. **Week of September 30: Review Sample Syllabi**

We will then review sample syllabi to identify common features and formatting. This will help us build a flexible template that our program can use as reference. This will include training a machine learning model to identify aspects of syllabi that may be slightly different in presentation, but adequate in content, such as labels like “How I Grade” substituting the ‘official’ “Grading Policy”.

1. **Week of October 7, 14: Build the First Version of the Program**

Using the template, we’ll create a program that can scan a syllabus and rate it based on its readability, whether it meets PSU Abington’s requirements, and overall user-friendliness.

**October 16:** Present project updates to the class.

1. **Week of October 21, 28: Make the Program Flexible**

After the initial version, we’ll tweak our program so that it can handle different syllabus formats, variations, and unusual cases such as pictures or special fonts.

1. **Week of November 6, 11: Add Content Checks and Recommendations**

Add the functionality where the program will flag missing our outdated sections and suggest improvements to make the syllabus more user-friendly for students.

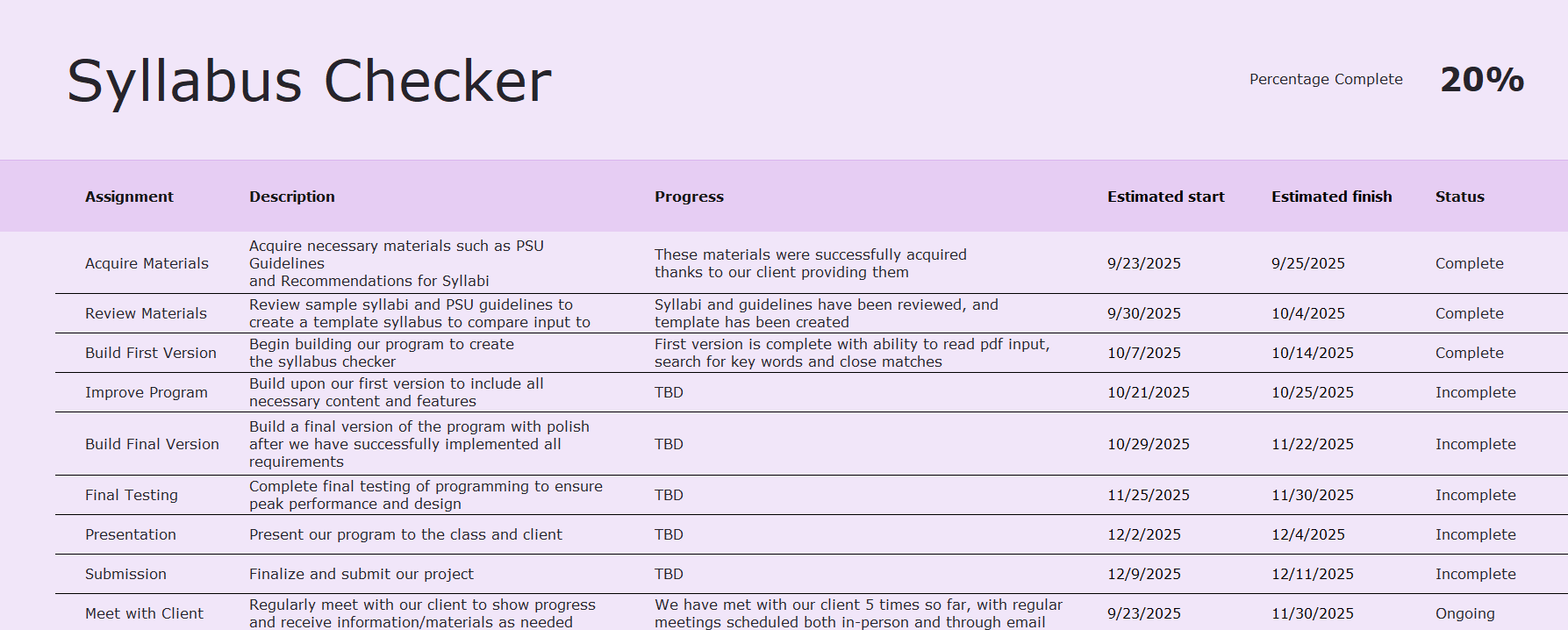
**November 6:** Present project updates to the class.

1. **Ongoing: Regular Check-ins with Client**

We plan on meeting with our client, Dr. Judy Ozment, every two to three weeks to ensure that we’re on the right track and that the project meets Dr. Ozment’s expectations.

**Final Testing and Presentation**

* Week of Dec 2, 4: Final Project Presentation
* Week of Dec 9, 11: Submit final project package, including:
  + Final Project Report (PDF)
  + PowerPoint Presentation Slides (PDF)
  + Evaluation Letter from Dr. Ozment
  + Source code or link to source code with instructions for running program



[Syllabus Checker Work Plan](https://pennstateoffice365-my.sharepoint.com/:x:/g/personal/jfd5838_psu_edu/EULT1djZkwhAh09H3dYWbZoBZPg-Li8uk3EME2EB5I7Zsg?e=WnXspZ)

**October 16th progress update:**

**PROJECT COMPLETION: 20%**

So far, we have implemented the functionality for reading and analyzing PDF syllabi files. Our program can now:

* **Scan and read PDF documents**: Using the pypdf library, the system extracts text from each page of a syllabus and compiles it into a single string for processing.
* **Identify course information:** The script automatically parses the filename to extract the course code, instructor name, and semester based on naming conventions.
* **Check for required content:** With the rapidfuzz library, the program performs keyword-based fuzzy matching to detect whether essential syllabus sections (such as email, grading, materials, office hours, etc.) are present. It then outputs a detailed “Content Analysis Report” and calculates an overall completeness score.
* **Perform semantic search:** Using the pre-trained CrossEncoder model from sentence-transformers, our system can semantically evaluate whether a user-entered query (e.g., “attendance policy”) appears in the syllabus text, which goes beyond keyword matching to detect meaning-based similarity. The output includes whether the queried content was Found or Missing, the model’s similarity score, and the best-matching sentence.

Our future plans include implementing a GUI where the user can easily upload a PDF and get the Content Analysis Report returned to them. Ideally, we would like to implement a set of hard-coded queries so that the program can automatically perform semantic searches for the various policies and guidelines that PSU Abington syllabi are required to have. We also plan to integrate a spell-check feature to help identify and flag spelling errors within the PDFs, ensuring that the analyzed syllabi are both complete and professionally written.

We continue meeting with our client, Dr. Ozment, with our next meeting scheduled for 10/20/2025. Our goal is to make sure we’re consistently meeting our client’s expectations and refining our project based on their feedback.

References

*Penn State Abington Faculty Handbook 2025 - 2026*

[*Syllabus Requirements*](https://senate.psu.edu/faculty/syllabus-requirements/)

[*43-00 Syllabi | University Faculty Senate*](https://senate.psu.edu/students/policies-and-rules-for-undergraduate-students/43-00-syllabi/)

*CMPSC 487W: Software Engineering and Design Syllabus*

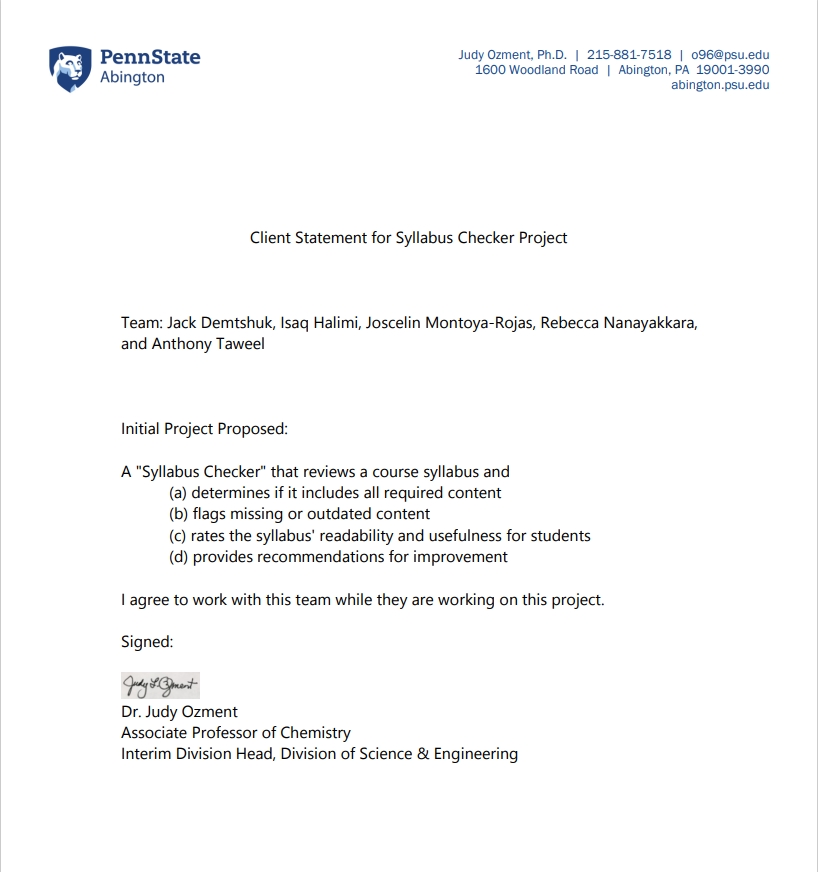
[*Grammarly: Free AI Writing Assistance*](https://www.grammarly.com/?msockid=2a3c34364751602b36ff224746c561c4)

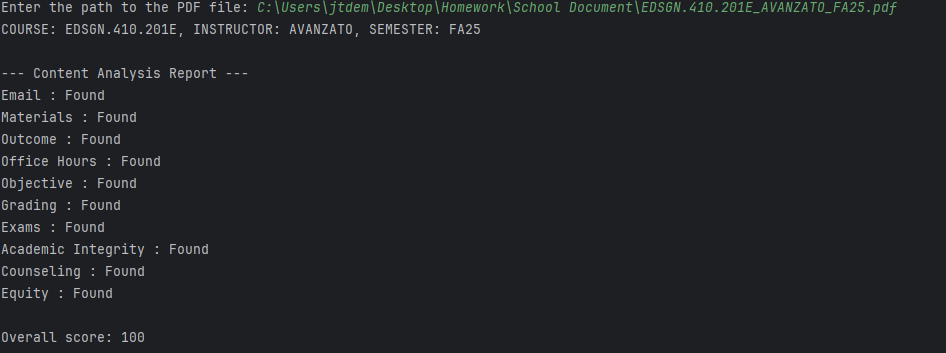
[*sentence-transformers · PyPI*](https://pypi.org/project/sentence-transformers/)

[*Pretrained Models — Sentence Transformers documentation*](https://sbert.net/docs/cross_encoder/pretrained_models.html)

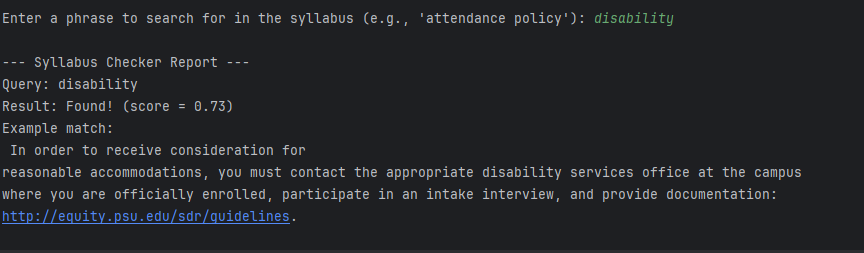
[*RapidFuzz · PyPI*](https://pypi.org/project/RapidFuzz/)

Appendix

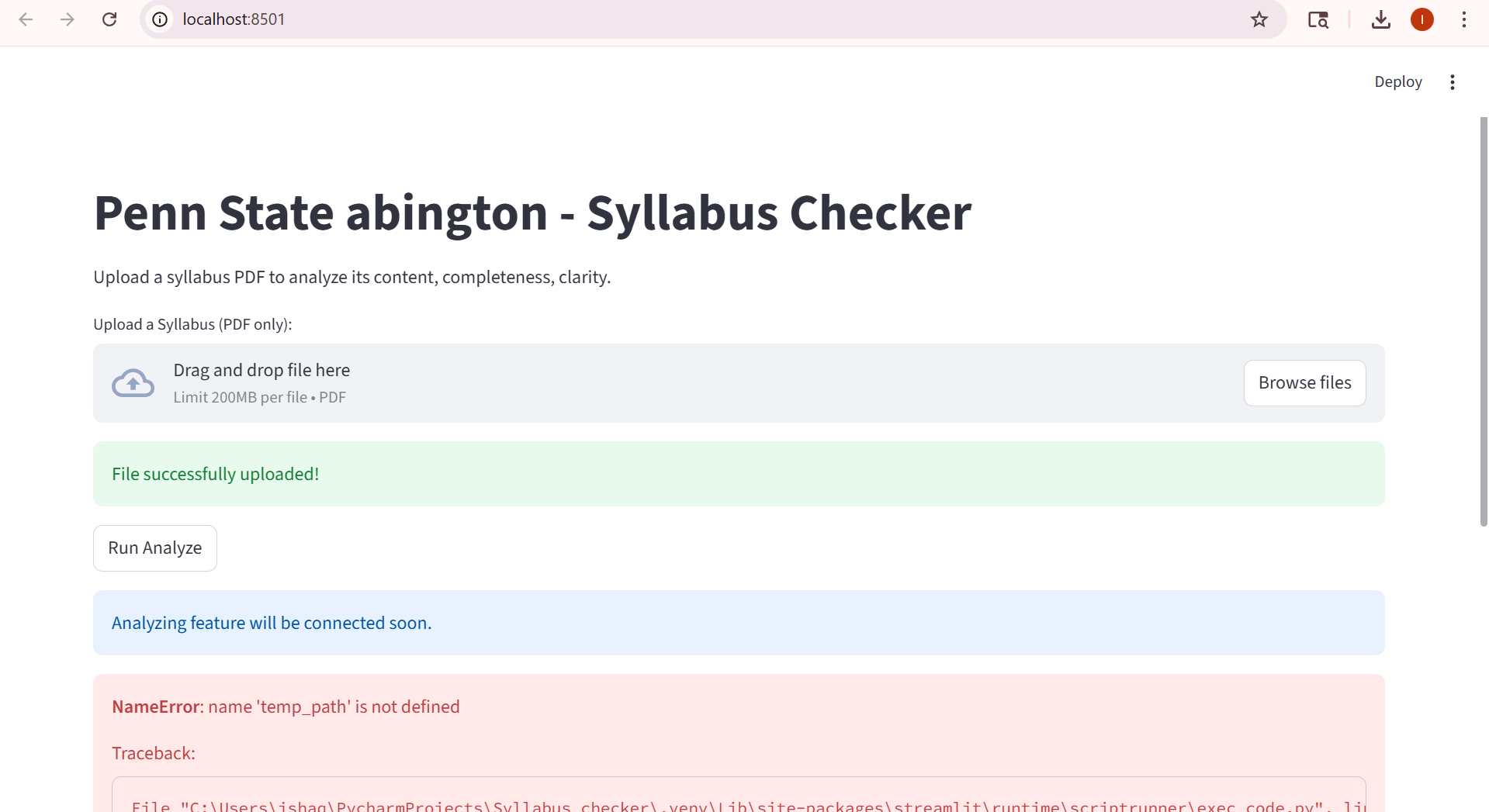




Screenshot displaying the input of a syllabus and the search of keywords to find if required content is present



Screenshot displaying a query that searches for best answer for required content



A Streamlit GUI that lets users upload a syllabus PDF and view analysis results.