TexIt

Project Manager: Bharath Srivatsan (bharaths@princeton.edu)
Other Team: Rohan Doshi (rkdoshi@prineton.edu) and Andrew Hartnett (arh3@princeton.edu)

I: Overview

Every week, students in engineering classes spend hours finishing their written assignments (aka "problem sets"). Since most students find it easier to handwrite their solutions to these mathematical problems, this means that they must waste multiple hours every week typing up their solutions into mathematical typesetting languages. The most popular of these is called LaTeX. To alleviate this inefficiency, we propose a web app that automatically generates LaTeX code from an uploaded image of a student's handwritten problem set.

II: Requirements and Target Audience:

User Needs:

University students from a variety of engineering and other technical backgrounds are assigned weekly problem sets. If already not a requirement, most professors expect or assume that students will LateX their assignments to ensure readability and consistent formatting. However, engineering students often write up and solve their problems sets on paper due to the ease of writing out mathematical notation. Thus, students must expend an additional 1-2 hours of tedious work per written page when retyping their work. Due to the complexity of the LateX language, students waste time constantly searching online for the LateX notation corresponding to various formatting schemes and mathematical symbols.

For students with non-technical backgrounds working on technical projects, papers, and psets, this problem is compounded. The added difficulty of writing correct, compilable LaTeX code can add further hours to this painful process. We aim to automate this complex and time intensive process of translating written problem sets to LaTeX files.

Existing Products:

There exist no other products on the market that translate problem sets from text to LaTeX in its entirety. The status quo is simply to manually retype written text into LaTeX. While the building blocks exist (e.g. computer vision APIs, optical character recognition via convolutional neural network, LaTeX recognition, content segmentation, etc.), piecing together these these technologies in a robust way is highly challenging, which is why nobody has yet successfully built a product for the proposed use case.

For example, this online demo from VisualObjects Web Equation demo demonstrates a simple app that allows users to write math text with their mouse and have it translated into LaTeX. However, this form of input is not a compelling use case since users would have to draw out their entire assignment with a computer mouse.

A slightly more compelling user flow was developed by Mathpix, which allows users to scan and upload individual formulas from a phone app. However, the difficulty and the time intensity of uploading a page's or a pset's worth of formulas one at a time renders this application irrelevant for the use cases we envision.

III: Functionality:

Use Case 1: Generating Assignment LaTeX From Image

After logging in and viewing the account dashboard, a user clicks the "Upload a New Assignment" button at the top of the page. The student will pick the picture of their problem set from the file selector.

This action will trigger a new view, with the pset image on the left of the screen and a button to "Generate LaTeX" on the right. The pset image on the left will have been pre-processed to enhance its contrast, sharpness, and alignment. Prompted by the instructions on the top of the screen, the user will click and drag boxes over the equations from the image. If the user wishes, they will be able to add images to the 'assignment' by clicking a button at the bottom left and selecting boxes for this new screen.

Finally, after clicking the "Generate LaTeX" button, the user will see on the right the output LaTeX code. From here, the user has the choice of editing it from within the browser, copying it to a different file, or downloading the code to submit elsewhere. As additional functionality, we hope to generate links based on this Tex code to ShareLaTeX documents, which will allow the user to edit and compile code directly using this third party service.

Use Case 2: Retrieving LaTeX Code From Past Assignments

The user wishes to edit his/her solution to a previous assignment. To do so, they navigate to our web app, login to their account, and glance at their account dashboard. The dashboard displays all of the student's past "assignments" line by line, in a style similar to Google Drive's file display. Users can click on any line to see their assignment. This entails full page screenshots on the left side of their screen, with arrows on the top to navigate between the pages of the assignment. On the right side of the page, the user can see the generated LaTeX code in a code editor. At the bottom of the screen, the user can see export options, such as the automatic generation of a ShareLaTeX link or a Tex file.

IV: Design:

Backend:

We propose a Flask framework to serve our site and handle our server-side endpoints. By choosing a python backend, it will be easier for us to leverage OpenCV and the other computer vision and machine learning APIs required in our project. Since we require asynchronous communication between the server and client, we will implement a Flask-RESTful API. We will heavily leverage:

- The OpenCV API for the preprocessing of our images (e.g. image alignment, cropping, contrasting, black-and-white coloring, etc.)
- The Google Cloud Vision API for optical character recognition and identification of blocks of text
- The Mathpix API to perform LaTeX character recognition from written equations

We will deploy our Flask app on an instance of AWS elastic beanstalk using a gunicorn server as our production-ready server.

Database:

We plan to use S3 to store individual image assets and tex files, in part because of its built-in connectivity with AWS Elastic Beanstalk. Furthermore, storing data-intensive assets on S3 will allow us to be cost-efficient, as its pricing plans (and free tier solutions) are comparatively generous. For the metadata involved in linking images with assignments and users, we plan to use MongoDB.

Frontend:

On the client-side, we will implement asynchronous HTTP requests and update our views via the React.js framework. We will implement jQuery to control UI interactions. As for account and session management, we will leverage the Flask-login library. Rather than creating a custom account management system (with usernames/passwords and associated change functionality), we plan to use OAuth v2 with Google Logins to register users and store relevant information.

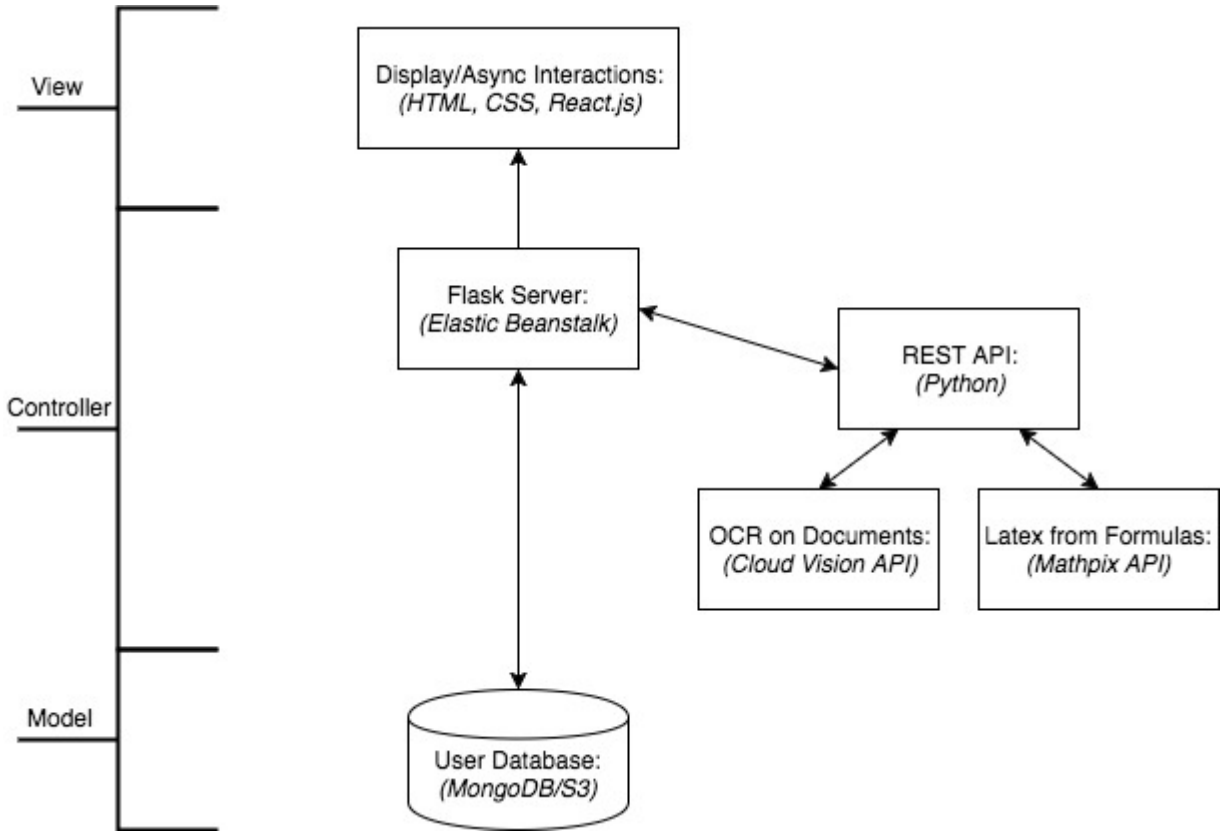

Figure 1: Proposed project architecture, along with MVC annotations

V: Timeline/Milestones:

We would like to have the following tasks completed by the following dates.

March 24th
We want to finish **prototyping the image-to-LaTeX asynchronous REST API**. By validating the core technology first, it will be easier to build a web platform on top later. This entails setting up the backend functionality for sending pieces and pages to the relevant apis, and stitching together the outcome pieces into a LaTeX document.

March 31st
We will finish the **second iteration of the image-to-LaTeX API**. At this point, we hope that we can ensure the robustness of the API under stress, and in various edge cases (for example, when the pictures aren't properly aligned in input or have poor user annotation of formulas).

April 7th
We aim to have an **unstylish, inelegant frontend done** by this date. There are a handful of technical challenges with the frontend (mostly the translating of user-drawn boxes into image coordinates), and so the frontend must be a priority early on. This version will allow a tester to upload an image and draw boxes on it. It will communicate with the API to send the data, but it will not handle receiving responses.

April 14th
A second version of the front end will **receive the response from the API**. The response from the API will be represented in both direct code form on the site and as a downloadable Tex file. This will be the official Alpha release.

April 21st
We will **implement user logins** by this date. The login will be associated with all of a user's past uploads.

April 28th
By this point, we will have implemented **better handling of headings, numbering, and graphs** within user uploaded images. We'll also have improved the front end, with a **prettier layout** of the upload, response, and user account pages. Finally, we'll build a splash landing page.

May 5
By this date, we hope to **develop a sense of "branding"** within the app (font, colors, personality), and receive user feedback on the relevant flows. We also will create instructions/support page.

May 12
We'll finalize the presentation-ready (beta) version of the app.

VI: Risks and Outcomes:

Our product relies on a lot of technological "moving-parts," such as computer vision and machine learning. Naturally, this introduces a few potential risks. Particularly salient are:

1. Our optical character recognition accuracy for handwritten text may be lower than expected.

Since a large portion of our project's functionality relies on our being able to adequately recognize and convert handwritten text to word processed text (and eventually to LaTeX), the possibility of misreading text could have a serious impact on user satisfaction. Given that we plan to rely on third-party APIs for the OCR portion of this project, we will mitigate the effects of this risk by building out the baseline REST API first, thereby allowing us to potentially switch to different OCR API providers if need be.

2. Our formula selection process may be inaccurate.

The decision to pass certain pieces through the MathPix API versus the cloud vision API is based off of users' selections of boxes from their input images. This process, though, could be prone to failure - users could select inaccurate boxes, those rectangle coordinates could be mistranslated in the cropping or API calls, etc. To mitigate this, we plan to experiment with fixed-ratio rectangles, and with automated box selection methods if our initial tests with user box selection don't pass.