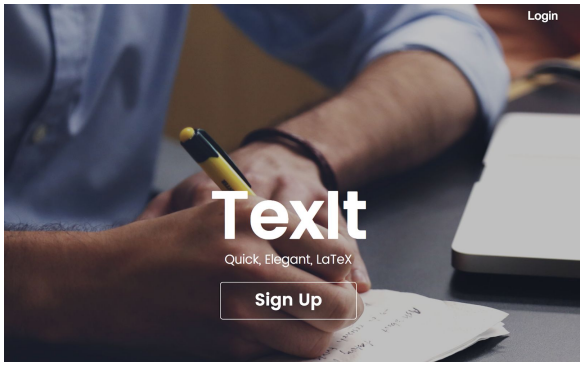
TexIt

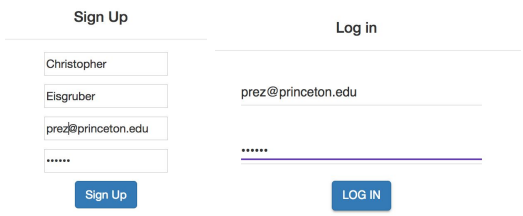
Product Guide

User Guide

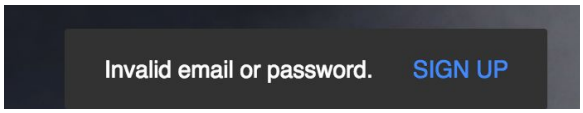
TexIt



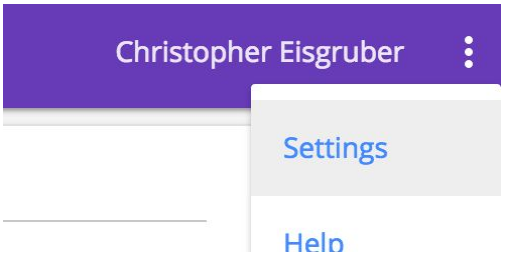
Landing Page



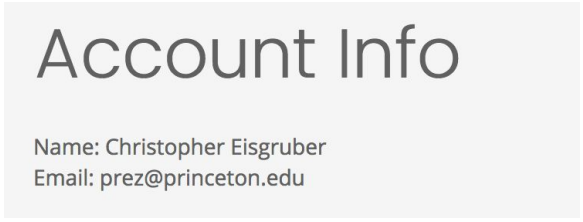
Sign up and login boxes



Error handling



Settings page



Account info on settings page

Creating your account

TexIt makes it easy for you to create and manage your account.

Account Creation

To create an account, you will need a valid email. Use this email to log in, along with your chosen password. Currently, we do not support changing passwords. Passwords must be between 3 and 40 characters.

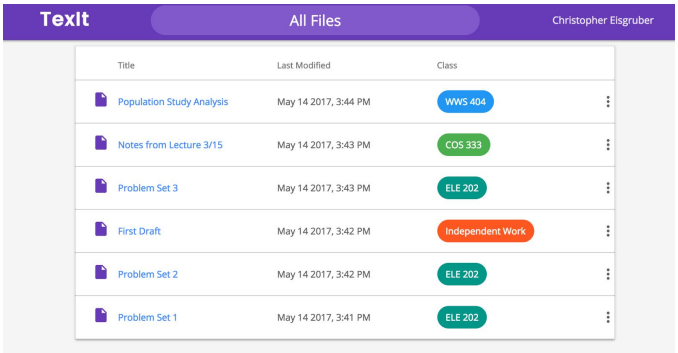
Using Your Account

After you login, you will see your name in the page header. By clicking on your name, or choosing "Settings" from the main menu (the dots on the top right of the header), you can see the rest of your account information.

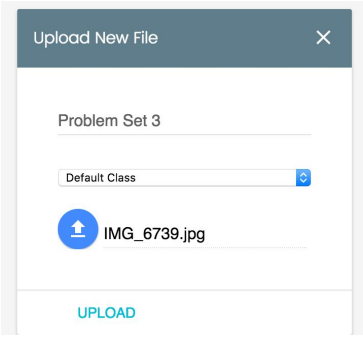
Your account is used to keep all your uploaded files together, along with the classes you use to keep them organized.

Learning More

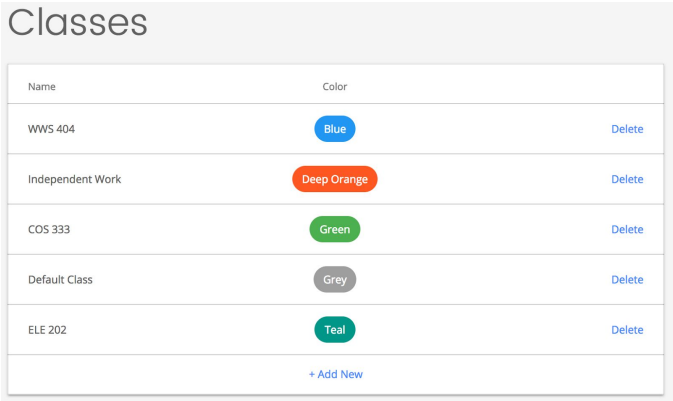
Also in the top menu, you'll see the options to be taken to a help page, for information about using our service, and an about page, where you can find information about the service and the people behind it.



Main files page

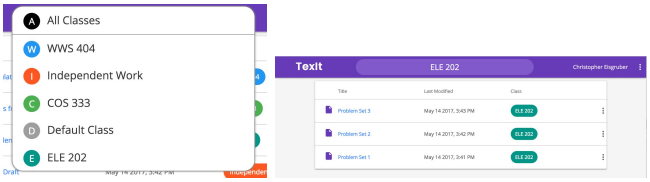


Upload form



List of classes on settings page

Creating a new class



Sorting by class

Managing your files

You can get started right away by uploading images and organizing your files.

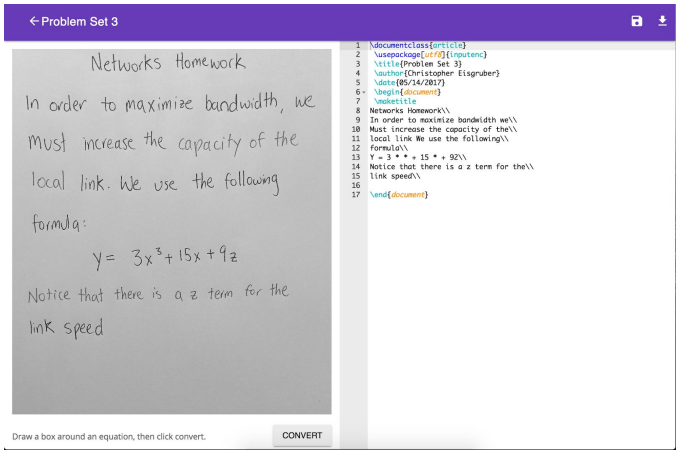
Uploading

Click the blue + button at the bottom right of the main page (click on the **TexIt** logo in the top left to get to the main page). In this form, you can create a title, select a class, and choose an image file to upload. We support .png and .jpg images, and recommend images no larger than 3200x3200 pixels.

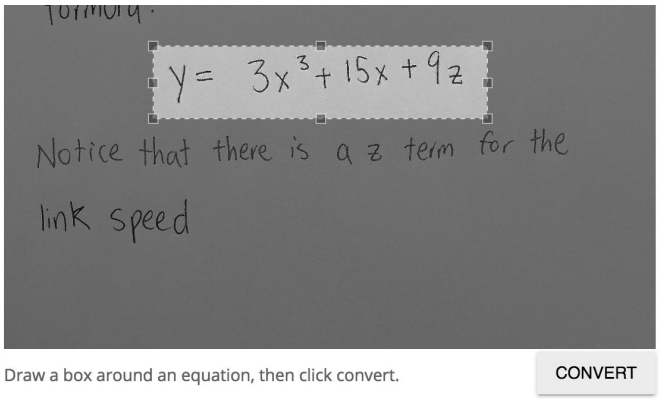
Organizing

Your files are all found on the main page. You can click on any one to view it's contents. The lefthand menu of each file allows you to delete the individual files.

Files can be sorted by "class". You can think of classes like folders, with assignments grouped together. From the settings page, it's easy to view, create, and delete classes (Be careful! Deleting a class also deletes the associated files). You must always have at least one class. When adding or removing classes, you may need to reload the page to see your changes.



Working with a single file



Selecting an equation

```
1  \documentclass{article}
2   \usepackage[utf8]{inputenc}
3   \title{Problem Set 3}
4   \author{Christopher Eisgruber}
5   \date{05/14/2017}
6   \begin{document}
7   \maketitle
8  Networks Homework\\
9  In order to maximize bandwidth we\\
10 Must increase the capacity of the\\
11 local link We use the following\\
12 formula\\
13 $$ y = 3x ^ { 3} + 15x + 9z  $$
14 Notice that there is a z term for the\\
15 link speed\\
16
17 \end{document}
```

Full LaTeX editor

Working with your files

When you upload a new file, or select one of your existing files, you can see and work with it's contents. Each file contains your original image and the LaTeX we pulled from it.

Viewing a file

When you look at the contents of a file, you'll see two panes. The left-hand pane is the image you uploaded and the right-hand pane is the LaTeX code for the text that your image contained.

The right hand view is a real code editor, with built-in LaTeX syntax highlighting. Go ahead, try it!

Equations

If part of your image contains equations, formulas, or other "math" content, you can convert these sections into LaTeX individually by clicking a dragging to create a box around the line in question, then clicking the convert button. The equation will then appear at the top of the content section of your code, and you can move it to the appropriate place in your document.

Saving and Downloading

In the header, you'll see two icons. Click the  icon to save the updated code to your files. Click the  icon to download a .txt file with all your code in it.

Coming soon: download a compiled .pdf version of your file!

Dev Guide

TexIt

This section outlines the architecture of the app, along with the key design considerations made to ensure its design is secure, robust, and scalable.

1. Overall Architecture

TexIt uses the Flask framework to implement a client-server architecture. The front end web client queries various server-side endpoints. The backend logic in application.py processes these requests by taking actions such as:

(1) Querying the MySQL database for user information
(2) Reading and writing files to the server-side file system
(3) Using the logic in api/run.py to interface with APIs (e.g. Microsoft's Cloud Vision API or Mathpix API).

Finally, the backend responds to the client with a dynamically rendered HTML template or a static HTML asset.

2. Frontend

2.1 UI/UX Components

To ensure cross-browser consistency and responsive design, this app leverages two popular frontend design frameworks:

(1) Twitter Bootstrap 3
(2) Google Material Design Lite

These libraries are particularly useful because they provide many pre-built UI components to enable rapid prototyping. They also allow for a wide range of CSS and JavaScript customizations.

2.2 Modules

2.2.1 Past Uploads & Course Filtering

This view is responsible for displaying all past uploads for a chosen course (filtering is set to "All Files" by default). Querying the backend for this view results in two acts of SQL filtering. First, the user ID is used to get a list of courses to populate the course filtering menu at the center of the navigation bar. Second, a list of relevant uploads are selected after filtering by user ID and course ID. (see Section 2.2.3).

2.2.2 Image Upload Form

Front end validation of the file upload menu ensures that the user enters a valid title, selects a course, and uploads an image of an allowed file type. The form takes a few seconds to process the uploaded image on the backend, which is why an upload animation is used to delineate to the user that processing is occurring.

2.2.3 Code Editor

The code editor is the open source ACE editor, from Cloud9, which supports syntax highlighting, navigating, line numbering, and current line highlighting. To populate the view,, the client queries the backend for the LaTeX code and upload image associated with an upload ID. The backend extracts a string from the latex file and formats it to include line breaks; this string is rendered directly into the HTML template for the Code Editor. This template is also rendered with the path to the server-side image.

When the user presses the "Save" button in the top right of the navigation menu, a JavaScript function sends the current version of the LaTeX code in the client editor to the server; this overrides the contents of the server's LaTeX file associated with this upload. When the user clicks the "Download" button in the top right of the navigation bar, the client queries the server for the latest version of the LaTeX text file associated with the upload.

2.2.4 Equation selection

The front-end equation selection interface uses the jCrop library to capture the associated image coordinates. These coordinates are sent to a server endpoint in order to rerender the view with the updated LaTeX code.

2.2.5 Course Creation/Deletion

Users can create courses on the settings page. Form validation ensures that users enter valid course names and colors. Upon pressing "Save," the client queries a server endpoint to insert the new course into the database's Course table.

Users can also delete courses. Pressing "Delete" next to the name of a class queries a server endpoint that deletes the corresponding course entry from the database's Courses table. The user cannot delete a course if there is only one left.

3. Backend

3.1 MySQL Database

We implement a MySQL database to store user information. We segmented user information using the three following tables:

(1) ***Users*** maps each user ID to other user account characteristics, such as first name, last name, email, and hashed password.
(1) ***Courses*** maps each course ID to a user ID, color, and course name.
(2) ***Uploads*** maps each upload ID to a user ID, class ID, latex file, image file, and upload date.

Each table entry has a unique primary key.

3.2 Image to LaTeX Conversion via APIs

Uploaded images are preprocessed with Pillow, an image preprocessing library written in Python, and sent to the Microsoft Cloud Vision for optical character recognition (OCR). The text is wrapped in LaTeX template code to produce a basic LaTeX document.

(1) ***Image Preprocessing:*** Pillow rotates the image so that text is facing upward, resizes the image to fit within the limits imposed by the OCR APIs, applies image filters to recast the image in greyscale, and enhances edges.
(2) ***Optical Character Recognition:*** We query the Microsoft Cloud Vision API for OCR of the pre-processed image.
(3) ***OCR LaTeX Formatting:*** The returned text is then wrapped with basic LaTeX syntax, such as headers and footers, to provide the bare minimum code for a functional LaTeX file.

(4) ***Equation LaTeX:*** If the user elects to use the equation selector, the relevant backend endpoint receives the image coordinates associated with the equation. Pillow crops the image based on the coordinates and feeds this image subset to the Mathpix API via a URI. The LaTeX for the equation is then returned from the API to the server, which is then rendered back into the code editor.

3.3 File Storage and Manipulation

Every time the user saves an updated version of the LaTeX code in the editor, the associated server-side text file is overwritten.

4. Security

4.1 Front end: Form Validation

We provide front end form validation for all input mechanisms to prevent the server from receiving unpredictable input types or attacks.

4.2 Backend: Access Control

During account creation, all passwords are stored in the database in a hashed and salted form. Server-side validation of passwords against the stored hashed version is done using the Flask Werkzeug Security library.

4.3 Backend: Session Authentication

Session cookies are set to maintain state during the time which a user is logged in. These session cookies are encrypted through the Flask Session library to ensure confidentiality and integrity of user information.

4.4 Backend: Endpoint Authorization

Server endpoints validate all GET and POST arguments to ensure that all request parameters are valid and not the product of attacks. For example, the server's /login endpoint validates a passed email by:

(1) Checking that string is formatted as a regular email address (with a "@" sign) using regular expressions.
(2) Checking that the email is between 3 and 40 characters.
(3) Checking for the email in the database.

4.5 Backend: Attack Prevention

Once the endpoint parameters have been validated, we must also prevent the possibility of a SQL injection attack. This is done by leveraging a Flask client for MySQL built on top of a library (PyMySQL) that automatically escapes all query variables.

5. Deployment

We heavily leverage Amazon Web Services (AWS) to deploy our web app as a series of microservices. We deploy the Flask app using AWS Elastic Beanstalk, which automatically provisions EC2 instances for hosting the app and S3 instances for server-side file storage (used for LaTeX text files and upload images). We deploy our MySQL tables through the AWS Relational Database Service (RDS).