Class Scribe: A Modern Approach to Note-taking

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Henry Weber
Spring, 2020

Technical Project Team Members
Benjamin Brown
Rahat Maini
John Watkins

Class Scribe: A Modern Approach to Note-taking

Abstract

Our project started out with the intention to provide students with a method of note-taking that allows students to access their notes anywhere with an internet connection, but maintain the benefits of pen on paper note-taking. Researchers have found that taking notes on pen and paper is the most effective way to take notes, even when compared to taking notes on a tablet with a stylus (Mueller & Oppenheimer, 2016). The system we designed is composed of multiple lamp-like devices (fitted with a Raspberry Pi) that will be positioned with a camera facing downward over each student's notebook. While in class, the lamps will take pictures of the notes on a set interval and also record the audio lecture; the data captured by the Raspberry Pi will be uploaded to the web app which is the second part of the system. The web app will be where students can go to view their personal notebooks and notebooks shared by other classmates. The web app is also used to configure the system which consists of the courses, lamp locations, and professors.

Our team relied on GitHub for version control, Slack to communicate outside of class, and in-person meetings to discuss project requirements and discuss how progress was going. We are looking forward to having our product embraced in the educational community as it provides the best of both worlds in terms of portability of notes and information retained from the actual act of taking notes. Recently students are mostly switching to entirely digital forms of note-taking (either typing or stylus on a touch screen), but these forms have led to declined academic performance and with Class Scribe this trend could be reversed.

1. Introduction

Currently there are three major ways of note taking in lecture halls: classic pencil and paper, laptop, or tablet with a stylus. Countless studies have assessed the ineffectiveness of typed notes for retention of material, and even more have laid out the distractibility of tablets (such as iPads) where notifications fight for your attention. Pencil and paper is an optimal writing experience, however it lacks the modern touch that makes digital note taking so lucrative (cloud syncing, search, no weight of books). Class Scribe is a project that aims to bring the best of modern digital note taking to traditional paper and pencil, with minimal management.

1.1 Problem Statement

It was clear to us that no major note-taking method is inherently bad. There are drawbacks and also positive points. With a laptop/tablet, we see that students become easily distracted but feel very at ease thanks to the ability to record lecture audio or type near the speed of speech. Not missing any key information is a strong benefit of methods that emphasize speed and efficiency in notation. With a smartpen, we see that students prefer the flexibility, ease, and intuitiveness of pen/paper but do not appreciate the added cost of specialized hardware and notebooks. The naturality of handwriting is well-known, with no fumbling of interfaces to manage content, and people undoubtedly (and perhaps unknowingly) appreciate such design for a tool as fundamental as the pen and paper. Google Docs and other note taking software has ushered in an era of digital collaboration and content management that has previously never existed, a notable achievement that propels forward the notions we have regarding work, note-taking, and study.

1.2 Contributions

Given the problem statement, we were able to construct a basic idea of a successful product. To meet our goals, we had to make sure the actual act of note-taking had to be unobstructed pen/paper notation. This was achieved by giving users the freedom of using their own writing utensil and paper, by developing a camera system for note-capture. We know capturing lecture audio is an important feature that brings ease to those who fear missing out on pertinent information, and we achieved this through a microphone system for audio-capture. Finally, we realize the importance of collaboration and portability of user content, for which we designed an entire web app to manage, share, and access your notes.

2. Related Work

There are a variety of systems that contend against Class Scribe using various faculties. Smartpen technology (such as that developed by Livescribe) is able to digitize handwriting, record and sync lecture audio to handwritten notes, and boasts many note-sharing options. Software such as OneNote and Evernote offer handwriting options for note-taking as well as audio capture and keyboard input, with free cloud syncing. Alternatives like Rocketbook, physical notebooks that contain infinitely erasable sheets of paper, offer a smartphone application where the user can scan their handwritten notes and save a digital copy to their chosen cloud storage provider. Finally, rounding up the multitude of options for note-takers are specialized note-taking tablets such as the reMarkable 2, which offers an e-paper display that resembles real ink on paper for a more natural note-taking experience, as well as handwriting digitization (increasingly appears to be a standard feature on many of these note-taking tools).

The aforementioned systems all aim to perform similar functions as Class Scribe and are highly specialized software/hardware offerings, however each system falls short either in execution or approach. Primarily for the specialized devices, offloading the cost to the user instead of the educational institution is a mistake. Selling users multi-hundred dollar devices designed for one purpose is a strategy that will ensure a standard is never achieved and improved upon. In fact, this is a strategy that has only ever worked once before in education and resulted in the monopoly of Texas Instruments in the calculator business. A $400 e-paper tablet, or a $100 smart pen incompatible with all notebooks save for the $20 ones the manufacturer sells, will never penetrate the student market to the degree that a free and, notably (for competition promotion) optional, note-taking device would.

A point of failure for non-paper/pen devices is their ineffectiveness to perfectly replicate the experience of pen/paper. This ineffectiveness stems from a multitude of reasons, one being the resultant latency between your device's stylus tip and the strokes drawn on screen. Recent advancements in relatively expensive technologies such as the 2018 iPad Pro have brought down this latency to ~20ms, an improvement however still perceptibly inferior to the physical tools it is trying to mimic (Miller, 2017). Another reason is the drawing parallax, in other words the distance between the digital ink and the tip of the stylus. With pen/paper, there is no gap between the two. There is no glass, digitizer, and optional screen protector between your work and the tool used to construct it.

Finally, by far the note-taking method that is gaining traction the fastest is also shown to be the most problematic in terms of actual memory retention and learning comprehension. Countless studies have shown the detriment of having access to a keyboard, how students turn to transcribing rather than summarizing and effectively learning through listening. Likewise, studies have shown the benefits of pen/paper note taking for cognition in comparison to typing out notes.

3. System Design

The goal of our system is to empower our users with useful features that vary among the different types of users. The users that will interact with the system can be categorized into three types: students, teachers, and admins. All user types will be able to register and log in to the web app, with students also being able to enroll their student ids with the lamp to tie their id to their user accounts. When students use the lamp while taking notes, the lamp will record lecture audio and periodically scan their notes before uploading them to the web app. In the web app, students will be able to access their individual notebooks and pages uploaded from the lamp, edit the name and privacy of their notebooks, delete their notebooks, view and rate the public notebooks of other students in their class, view their scanned notes from lecture, listen to lecture audio, sync the audio's current time to a page or sync the current page to a specific point in the audio, export a page as a pdf, and send their page to their professor. In the web app, professors will be able to access notebooks for the classes they are teaching containing pages sent to them from students, endorse public notebooks for use by students accommodated by the Student Disability Access Center, and export pages as pdfs. Admin users will be able to create new course entries and edit existing course entries; able to set the name, room number, meeting times, building, professor identifier, master lamp serial number, and semester associated with the course course.

To build the application, our team decided to use Django for the backend, and React for the frontend. We chose Django for the backend because we were all familiar with the framework from prior coursework in the CS major. While we could have built the entire app in Django, we decided to use React to build the frontend, due to the greater functionality it provides over Django when building responsive user interfaces. Our code is licensed under General Public License v3.0, allowing commercial use, modification, distribution, patent use, and private use of our code.

3.1 System Requirements

Gathering system requirements is a key part of the development process. This step is so crucial because requirements provide a basis for what the customer expects to be delivered, which can then be used by the development team to make sure all needs are met. Without gathering requirements, there would be no assurance that what we were building would satisfy the customer’s needs.

The following are our minimum requirements in the form of user stories:

- As a student, I will be able to sign up for a Class Scribe account through the web app
- As a student, I will be able to enroll my ID to my Class Scribe account through Lamp
- As a student, I will be able to sign into my Class Scribe account on Lamp (after ID enrollment) and on the web app
- As a student, once I sign into Lamp I will have it scan my notes and record the lecture audio around me
- As a student, I will be able to see my scanned notes on the web app, hear lecture audio, and read the transcription of that audio
- As an administrator, I will be able to assign a class, classroom, meeting time, and lamp serial number through the web app
- As an administrator, I can view all of the course assignments for the selected room.

The following are our desired requirements in the form of user stories:

- As a user, I will be able to toggle dark and light mode for the web app.
- As a student, I will be able to sync my page to the current time in audio playback.
- As a student, I will be able to rename each of my notebooks.

• As a student, I will be able to sync the place in the audio playback to a specific snapshot in a page.

• As a student, I will be able to set my notebook as public for others in my class to see.

• As a student, I will be able to find other students’ public notebooks for use as a study material.

• As a student, I will be able to export an entire notebook as a pdf.

• As a student, I will be able to enroll my ID with the lamp.

• As a student, I will be able to push a button on the lamp to signal a page switch, to ensure my next scan is placed in the correct page.

• As a professor, I will be able to label my students' notebooks as SDAC ready.

The following are our optional requirements in the form of user stories:

• As a student, I will be able to toggle between the transcript of the lecture audio and the transcript of my scanned notes (OCR).

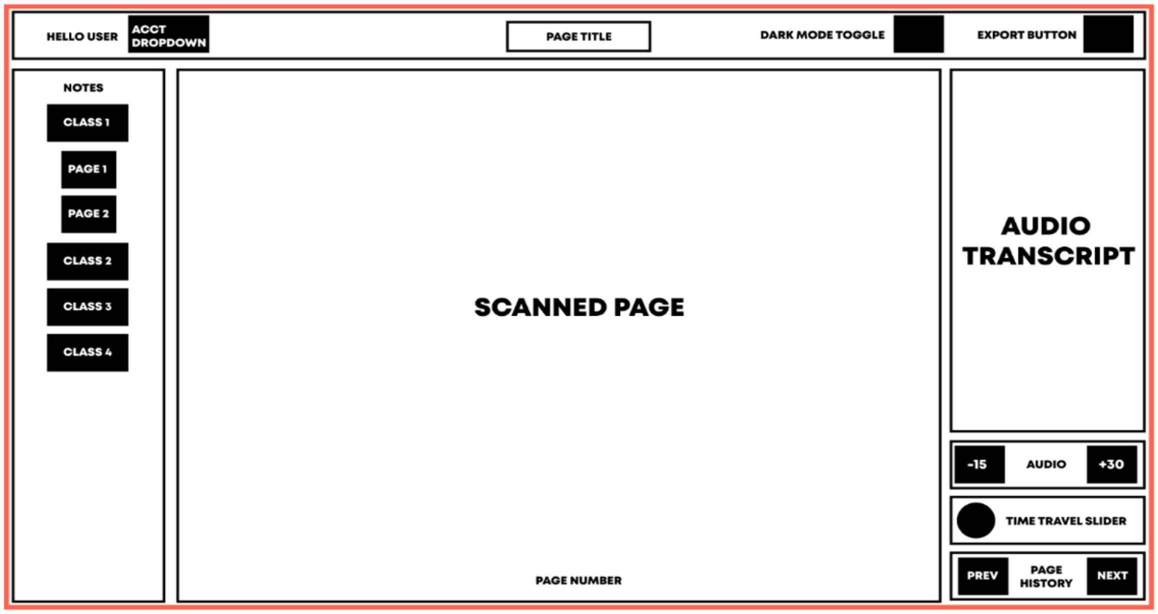
• As a student, when I switch pages while taking notes, the lamp will recognize this change and place the next scan in the correct page.

• As a professor, I will be able to see my classes attendance through the web app.

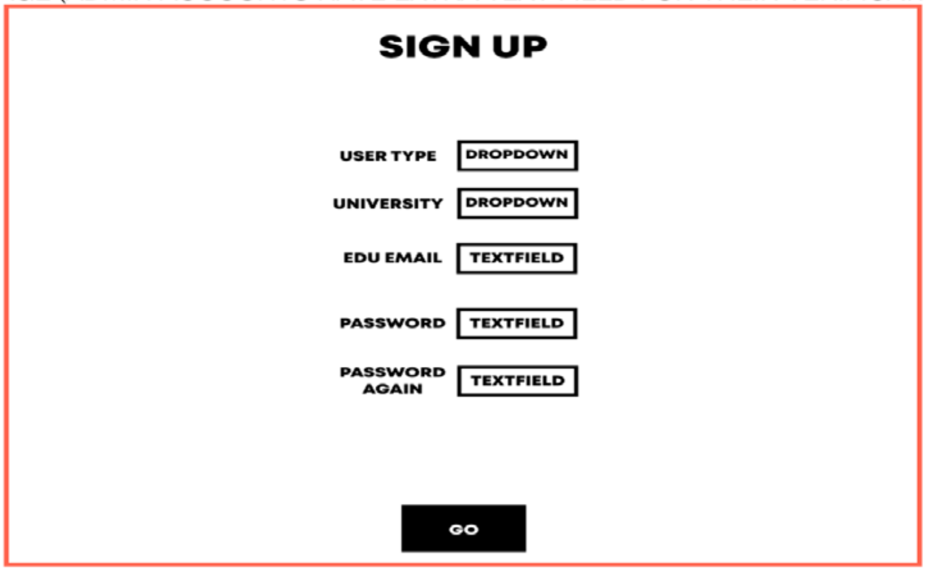
3.2 Wireframes

For our project, wireframes served several key purposes. When meeting with the customer, having wireframes allowed our team to give the customer a clear idea of the structure of our applications interface. This allowed the customer to provide feedback to make sure the final result met their expectations. Wireframes were just as critical for our team as developers, allowing us to develop and critique our vision for the applications interface before writing the code. This made the development process more efficient, minimizing design changes while developing the application. Our wireframes are provided below.

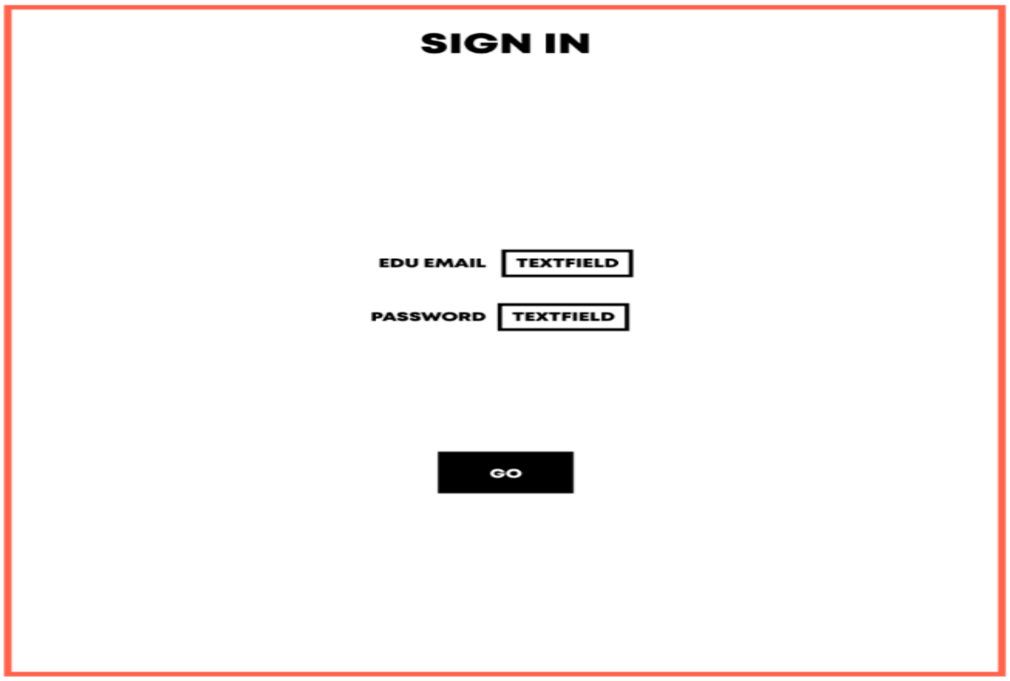
MAIN WEBAPP FOR STUDENTS



SIGNUP PAGE (ADMIN ACCOUNTS HAVE EXTRA TEXT FIELD FOR THEIR VERIFICATION CODE)

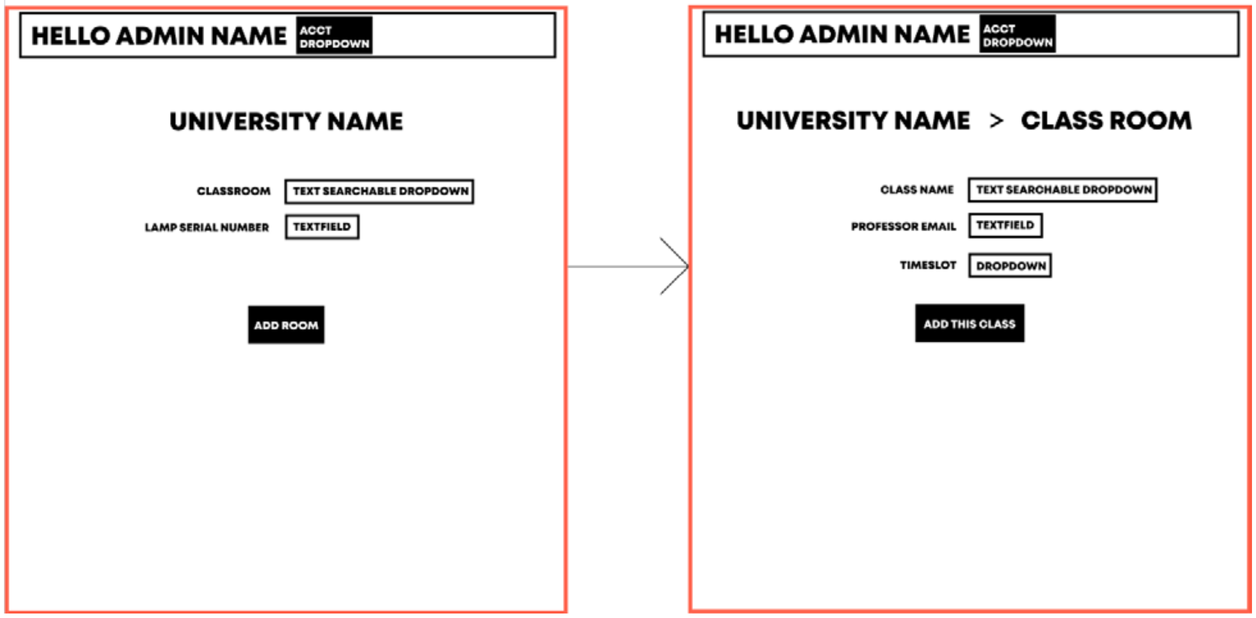


SIGN IN PAGE

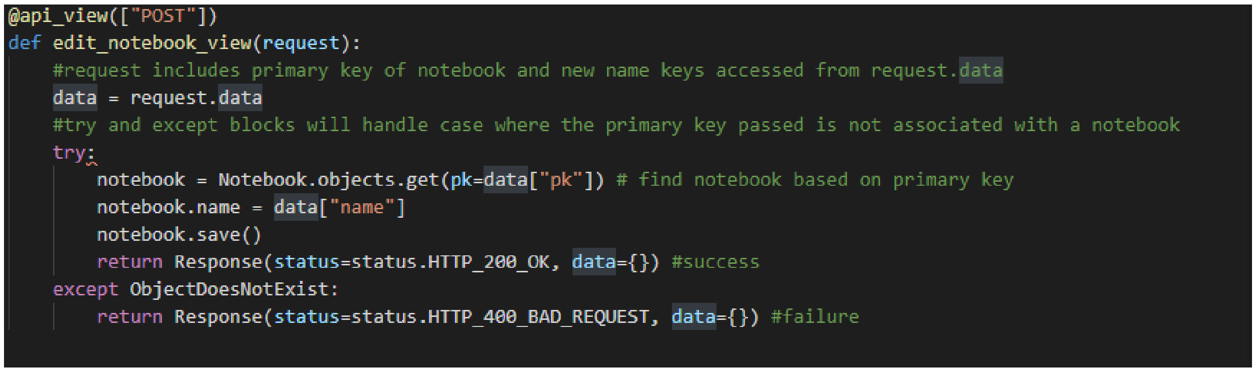


ADMIN SCHEDULE CREATION

ADMIN SCHEDULE CREATION

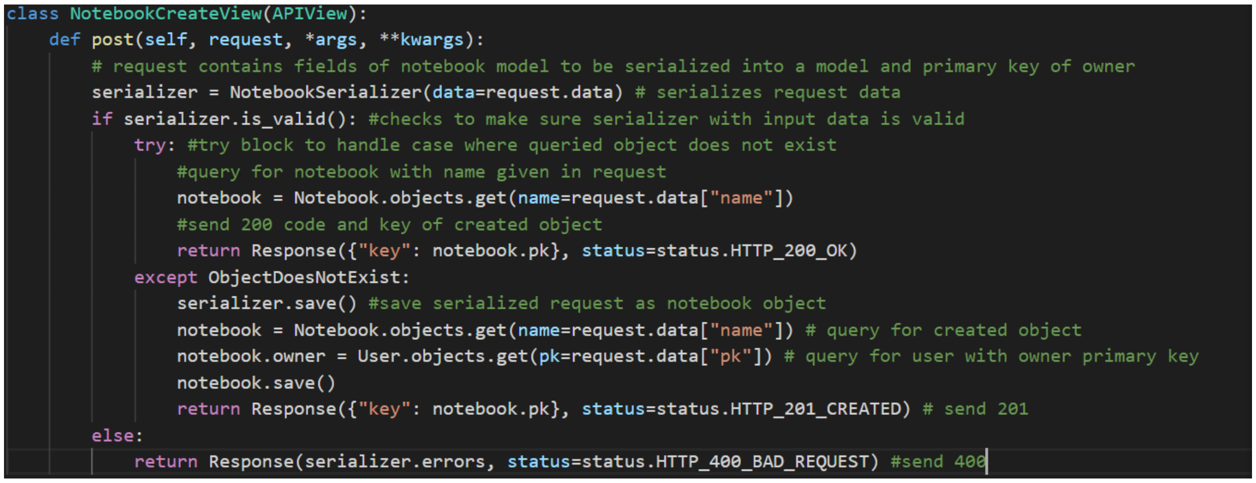


3.3 Sample Code



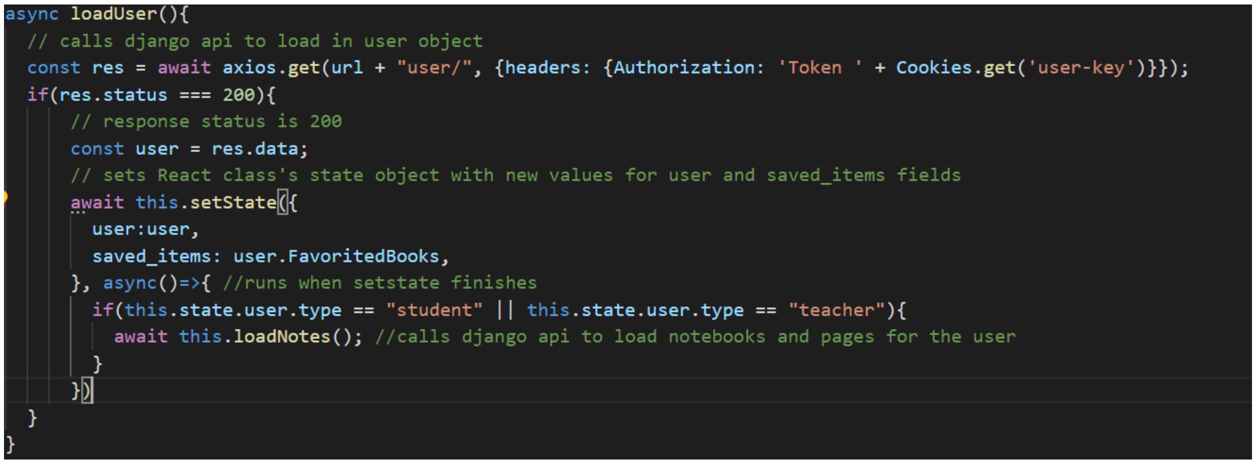
```python
@api_view(["POST"])
def edit_notebook_view(request):
    #request includes primary key of notebook and new name keys accessed from request.data
    data = request.data
    #try and except blocks will handle case where the primary key passed is not associated with a notebook
    try:
        notebook = Notebook.objects.get(pk=data["pk"]) # find notebook based on primary key
        notebook.name = data["name"]
        notebook.save()
        return Response(status=status.HTTP_200_OK, data={}) #success
    except ObjectDoesNotExist:
        return Response(status=status.HTTP_400_BAD_REQUEST, data={}) #failure
```

Figure 3.3.1: edit_notebook_view(Django) function that changes the name of a notebook based on input.



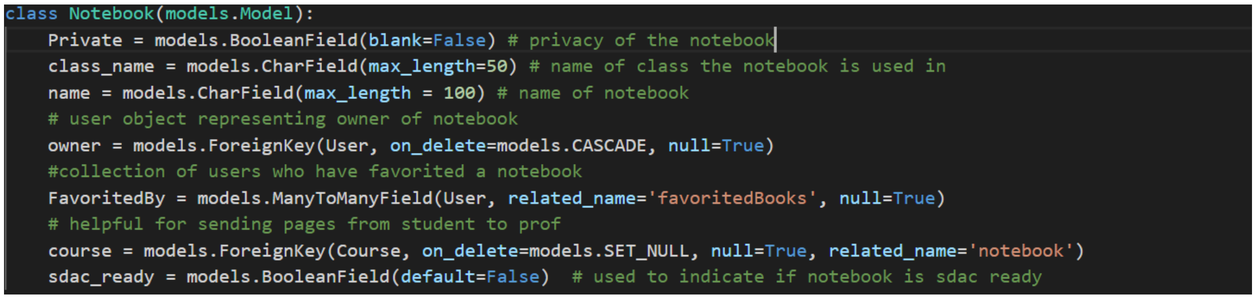
```python
class NotebookCreateView(APIView):
    def post(self, request, *args, **kwargs):
        # request contains fields of notebook model to be serialized into a model and primary key of owner
        serializer = NotebookSerializer(data=request.data) # serializes request data
        if serializer.is_valid(): #checks to make sure serializer with input data is valid
            try: #try block to handle case where queried object does not exist
                #query for notebook with name given in request
                notebook = Notebook.objects.get(name=request.data["name"])
                #send 200 code and key of created object
                return Response({"key": notebook.pk}, status=status.HTTP_200_OK)
            except ObjectDoesNotExist:
                serializer.save() #save serialized request as notebook object
                notebook = Notebook.objects.get(name=request.data["name"]) # query for created object
                notebook.owner = User.objects.get(pk=request.data["pk"]) # query for user with owner primary key
                notebook.save()
                return Response({"key": notebook.pk}, status=status.HTTP_201_CREATED) # send 201
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST) #send 400
```

Figure 3.3.2: NotebookCreateView(Django) function that creates a new notebook object and returns the key of the new notebook. If a notebook already exists with same name then it just returns the key of that object.



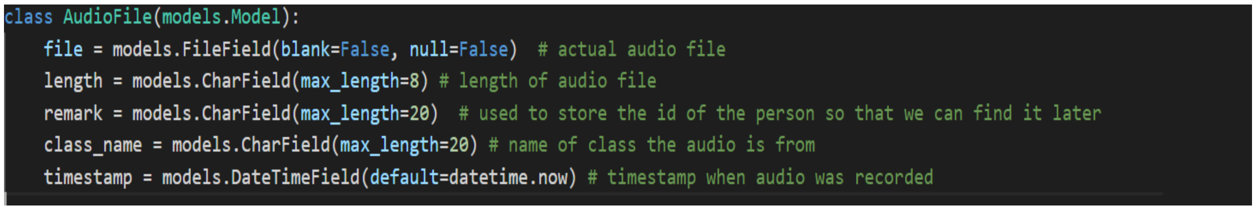
```javascript
async loadUser(){
  // calls django api to load in user object
  const res = await axios.get(url + "user/", {headers: {Authorization: 'Token ' + Cookies.get('user-key')}});
  if(res.status === 200){
      // response status is 200
      const user = res.data;
      // sets React class's state object with new values for user and saved_items fields
      await this.setState({
        user:user,
        saved_items: user.FavoritedBooks,
      }, async()=>{ //runs when setstate finishes
        if(this.state.user.type == "student" || this.state.user.type == "teacher"){
          await this.loadNotes(); //calls django api to load notebooks and pages for the user
        }
      })
  }
}
```

Figure 3.3.3: loadUser(React JS) function that requests user object from Django backend.



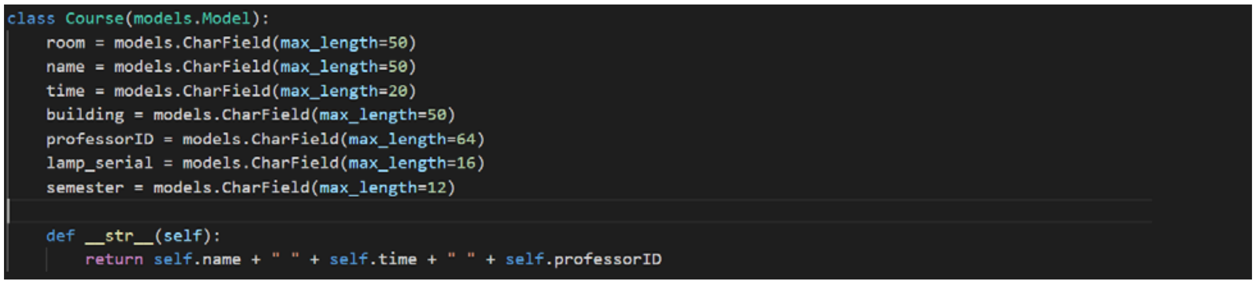
```python
class Notebook(models.Model):
    Private = models.BooleanField(blank=False) # privacy of the notebook
    class_name = models.CharField(max_length=50) # name of class the notebook is used in
    name = models.CharField(max_length = 100) # name of notebook
    # user object representing owner of notebook
    owner = models.ForeignKey(User, on_delete=models.CASCADE, null=True)
    #collection of users who have favorited a notebook
    FavoritedBy = models.ManyToManyField(User, related_name='favoritedBooks', null=True)
    # helpful for sending pages from student to prof
    course = models.ForeignKey(Course, on_delete=models.SET_NULL, null=True, related_name='notebook')
    sdac_ready = models.BooleanField(default=False)  # used to indicate if notebook is sdac ready
```

Figure 3.3.4: Notebook model class (Django), describing the structure of notebook objects used in the web app.



```python
class AudioFile(models.Model):
    file = models.FileField(blank=False, null=False)  # actual audio file
    length = models.CharField(max_length=8) # length of audio file
    remark = models.CharField(max_length=20)  # used to store the id of the person so that we can find it later
    class_name = models.CharField(max_length=20) # name of class the audio is from
    timestamp = models.DateTimeField(default=datetime.now) # timestamp when audio was recorded
```

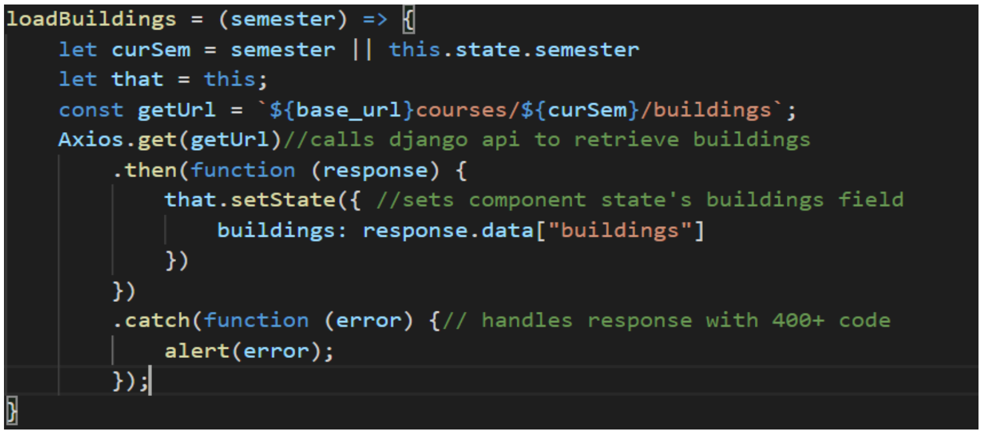
Figure 3.3.5: AudioFile model class (Django), describing the structure of AudioFile objects used in our web app.



```python
class Course(models.Model):
    room = models.CharField(max_length=50)
    name = models.CharField(max_length=50)
    time = models.CharField(max_length=20)
    building = models.CharField(max_length=50)
    professorID = models.CharField(max_length=64)
    lamp_serial = models.CharField(max_length=16)
    semester = models.CharField(max_length=12)

    def __str__(self):
        return self.name + " " + self.time + " " + self.professorID
```

Figure 3.3.6: Course model class(Django), describing the structure of course objects used in the web app. The class includes a __str__ function allowing the object to be displayed as a string.

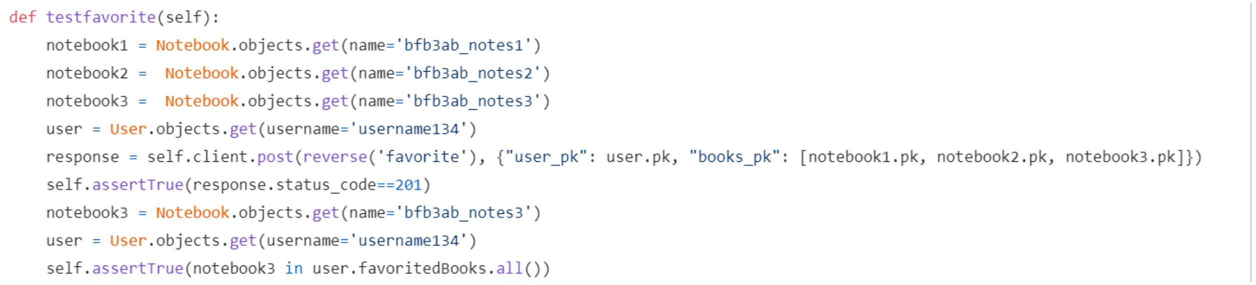


```javascript
loadBuildings = (semester) => {
    let curSem = semester || this.state.semester
    let that = this;
    const getUrl = `${base_url}courses/${curSem}/buildings`;
    Axios.get(getUrl)//calls django api to retrieve buildings
        .then(function (response) {
            that.setState({ //sets component state's buildings field
                buildings: response.data["buildings"]
            })
        })
        .catch(function (error) {// handles response with 400+ code
            alert(error);
        });
}
```

Figure 3.3.7: loadBuildings function (React JS) - function that loads buildings for a given semester and stores it in CourseCalendar component.

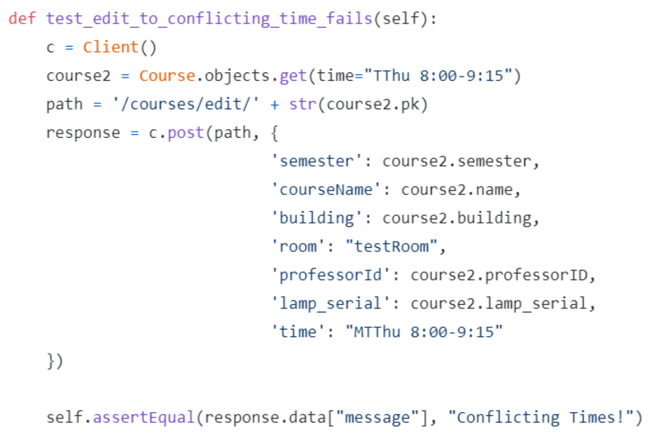
3.4 Sample Tests

Testing is a crucial part of the development process because without thorough testing, it is impossible to verify that system requirements have actually been met. Testing is also very useful because it often alerts developers to strange behavior exhibited by their code. These two major benefits of testing make it a critical part of development. To test our application we used Django’s test suite for the backend and Jest and Enzyme to test the frontend. Some sample tests are provided by our team below.



```python
def testfavorite(self):
    notebook1 = Notebook.objects.get(name='bfb3ab_notes1')
    notebook2 =  Notebook.objects.get(name='bfb3ab_notes2')
    notebook3 =  Notebook.objects.get(name='bfb3ab_notes3')
    user = User.objects.get(username='username134')
    response = self.client.post(reverse('favorite'), {"user_pk": user.pk, "books_pk": [notebook1.pk, notebook2.pk, notebook3.pk]})
    self.assertTrue(response.status_code==201)
    notebook3 = Notebook.objects.get(name='bfb3ab_notes3')
    user = User.objects.get(username='username134')
    self.assertTrue(notebook3 in user.favoritedBooks.all())
```

Figure 3.4.1: (Django) Test verifies the requirement that users should be able to favorite public notebooks to add them to their collection of saved notebooks or unfavorite them to remove a notebook from their collection.



```python
def test_edit_to_conflicting_time_fails(self):
    c = Client()
    course2 = Course.objects.get(time="TThu 8:00-9:15")
    path = '/courses/edit/' + str(course2.pk)
    response = c.post(path, {
                            'semester': course2.semester,
                            'courseName': course2.name,
                            'building': course2.building,
                            'room': "testRoom",
                            'professorId': course2.professorID,
                            'lamp_serial': course2.lamp_serial,
                            'time': "MTThu 8:00-9:15"
    })

    self.assertEqual(response.data["message"], "Conflicting Times!")
```

Figure 3.4.2: (Django) Above test tries to edit a course object so that its time will conflict with another course. The test is verifying that the action will result in an error and will respond with a specific error code, indicating the conflict.

```
it('popup renders and closed', async(done)=>{
  var mock = new MockAdapter(axios);
  mock.onPost(new RegExp('/favorite/')).replyOnce(400, {})
  let wrapper = Enzyme.mount(<NotebookCard
    parent={state}
    notes={notes}
    note={note}
    onUpdateUser={()=>{}}
    onUpdatePublic={()=>{}}
  />, {attachTo: container})
  let popup_button = wrapper.find('#ex').at(4)
  popup_button.simulate('click')
  //popup submit button
  let submitFavs = wrapper.update().find('#submitFavorite').at(4)
  await submitFavs.simulate('click')
  setTimeout(()=>{expect(wrapper.update().find('#submitFavorite').length).toBe(0);
    done()}, 400) //wait for popup close
})
```

Figure 3.4.3: (React) Above test verifies that the popup can be activated from the NotebookCard component and then verifies that it closes after a delay.

```
it("event renders with class info", () => {
    act(() => {
        render(<ClassEvent start={moment({h: 8, m: 0}).day(1)}
                           end={moment({h: 12, m: 0}).day(1)}
                           name="Test Class"
                           professor="Test Professor"
                           lamp="lamp"
                           days="MWF"
                           building="Test Building"
                           room="Test Room"
                           semester="Test semester"/>,
                container);
    });
    expect(container.textContent).toBe("8:00 - 12:00Test ClassTest Professor");
});
```

Figure 3.4.4: (React) The above test verifies that an event component renders correctly based on input provided.

3.5 Code Coverage

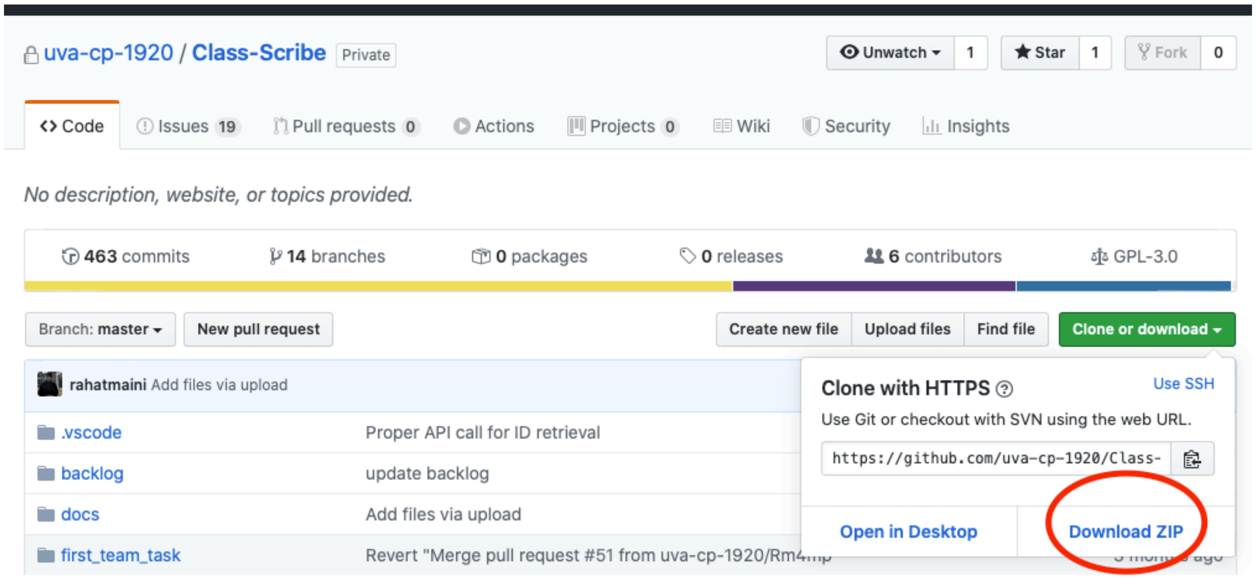
To measure code coverage for our project, we used coverage.py for the python backend, and istanbul for the javascript frontend. To set up coverage.py for measuring coverage, we first installed the package with the command **pip install coverage**. After the package was successfully installed, we were able to generate coverage reports with the package using the commands in the base django app directory; **coverage run --source="." manage.py test .** followed by **coverage html**, where the results were placed in the same directory the command was run in. The Istanbul coverage tool was provided in the **react-scripts** collection of scripts included when creating a React app with Create React App library. To generate code coverage reports with istanbul, we used the command **react-scripts test --coverage**.

When generating coverage reports with coverage.py, the reports include results for all files within the directory the command was run, indicating the name of the file, the number of statements, the number of uncovered statements, the percentage covered, and the line numbers of uncovered statements. Generating coverage reports with istanbul produces a similar report to coverage.py, however, for istanbul the metrics reported are percentage of statements covered, percentage of branches covered, percentage of functions covered, percentage of lines covered, and the line number of uncovered statements for each file in the directory.

3.6 Installation Instructions

Before being able to use the application, there are a few steps that must be completed first.

1. Download Heroku cli, git, and node:
    - **a.** Heroku cli: https://devcenter.heroku.com/articles/heroku-cli#download-and-install
    - **b.** git: https://git-scm.com/downloads
    - **c.** Node: https://nodejs.org/en/download/
2. Next download and unzip the codebase from the Class SCribe GitHub repo:
    - **a.** Link: https://github.com/uva-cp-1920/Class-Scribe/tree/master


uva-cp-1920 / Class-Scribe Private
Unwatch 1 Star 1 Fork 0
Code Issues 19 Pull requests 0 Actions Projects 0 Wiki Security Insights

No description, website, or topics provided.

463 commits | 14 branches | 0 packages | 0 releases | 6 contributors | GPL-3.0

Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download

rahatmaini Add files via upload

.vscode	Proper API call for ID retrieval
backlog	update backlog
docs	Add files via upload
first_team_task	Revert "Merge pull request #51 from uva-cp-1920/Rm4...

Clone with HTTPS

Use SSH

Use Git or checkout with SVN using the web URL.

```
https://github.com/uva-cp-1920/Class-
```

3. Open a console on your computer (terminal for MacOS, powershell for Windows).

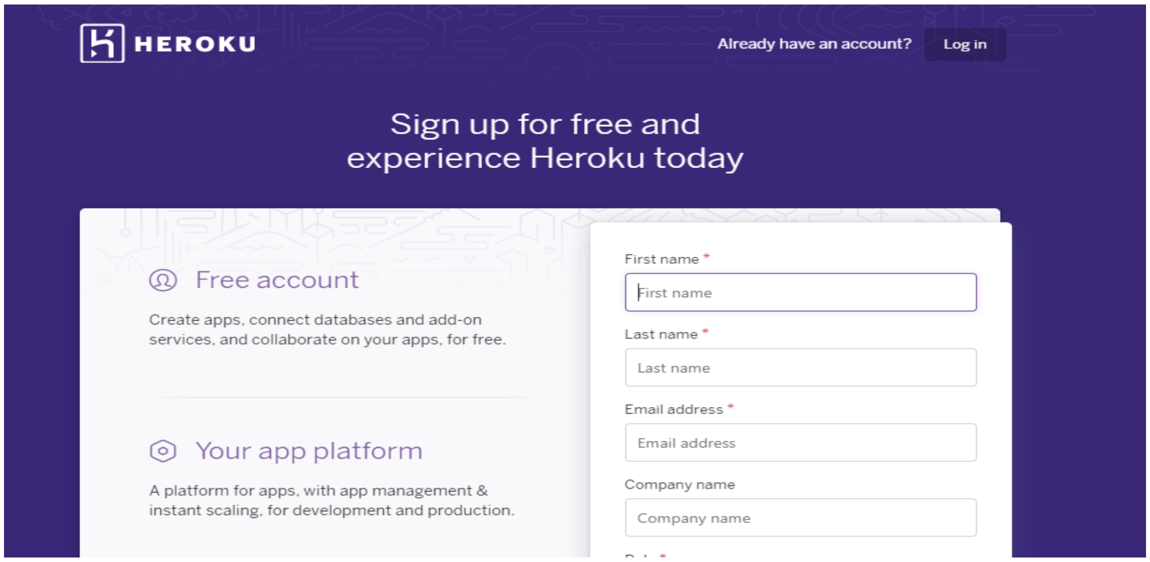
4. Use the terminal to navigate to the folder of the codebase you just downloaded.

    a. Can use command **cd <path of codebase>**

5. Go to this link to sign up for a Heroku account:

    a. https://signup.heroku.com

    b. After following the link you should see this view.



Sign up for free and experience Heroku today

Free account

Create apps, connect databases and add-on services, and collaborate on your apps, for free.

Your app platform

A platform for apps, with app management & instant scaling, for development and production.

First name *

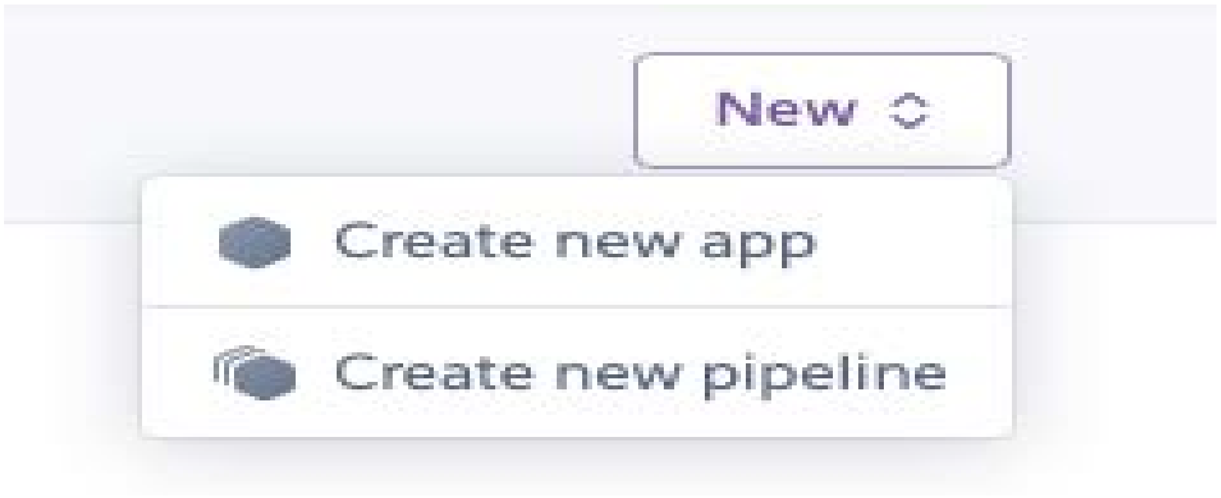
Last name *

Email address *

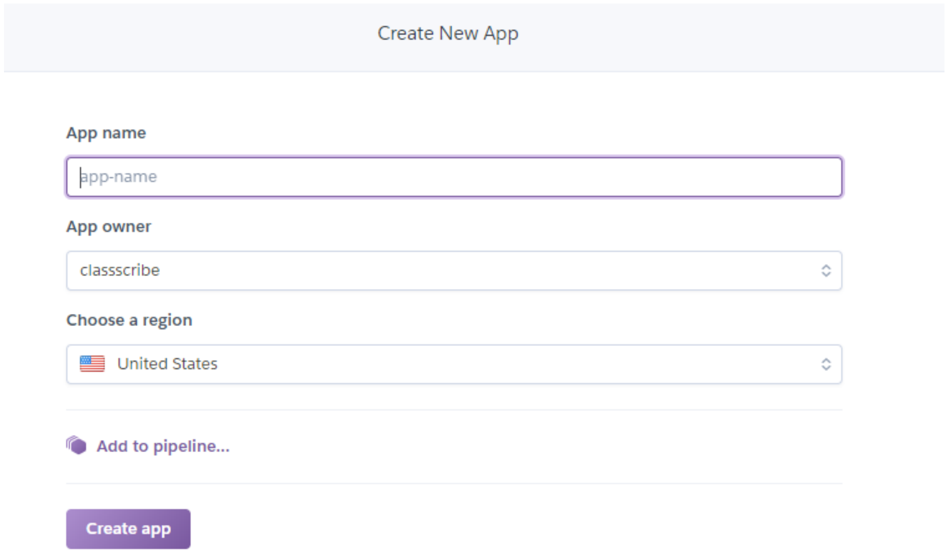
Company name

    c. You can then follow the prompts to signup.

6. After signing up for heroku, create two Heroku apps in your dashboard in Heroku (one for front-end, one for back-end) using the names of your choice
   a. You should see a button (text: **New**) in the top right corner of the screen.
   b. Click the button and you should see the following on your screen.



   c. Now click the create new app button.
   d. You should now see a screen that looks like this.


Create New App

App name

app-name

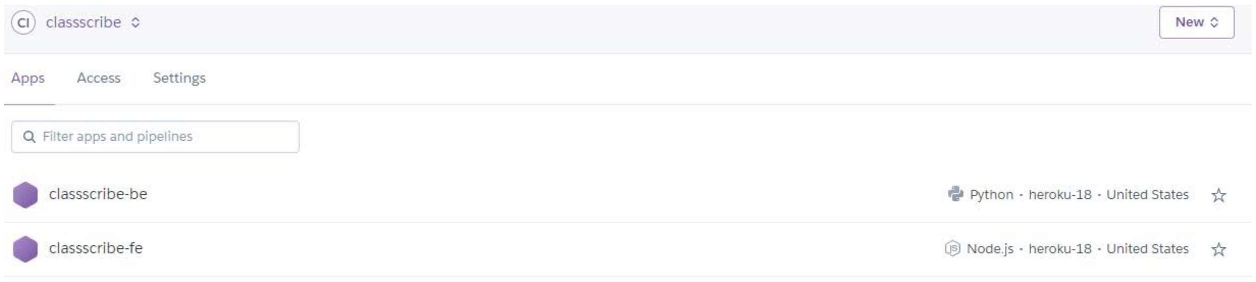
App owner

classscribe

Choose a region

United States

   e. Proceed to create both your frontend and backend apps.
   f. After returning to the dashboard you should see a view similar to this.



classscribe-be · Python · heroku-18 · United States

classscribe-fe · Node.js · heroku-18 · United States

7. Now click on the backend app in your dashboard and follow the next few steps:

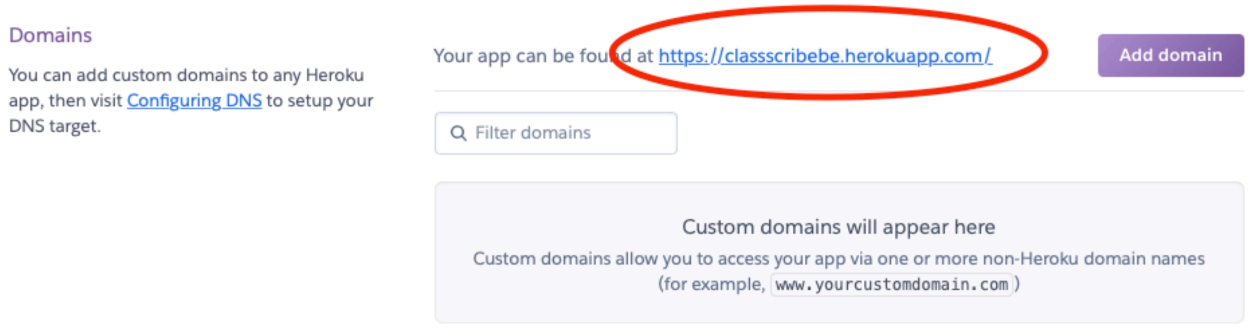
   a. Go to the settings tab.

   b. Under the buildpacks section, select "Add buildpack" and select Python

   c. Open your file explorer and navigate to where the codebase was downloaded to (named Class-Scribe-master)

   d. Go into the src folder, then the classscribe folder, then go to the api folder, then open the file called views.py using a text editor

   e. in the text editor, look a few lines down from the top and edit the line that starts with "server_url", take the url for your heroku app (for example https://classscribe-be.herokuapp.com/) in between the quotation marks on that line, and make sure to include a forward slash at the end of url when you paste it.



You can add custom domains to any Heroku app, then visit Configuring DNS to setup your DNS target.

Your app can be found at https://classscribebe.herokuapp.com/

Custom domains will appear here

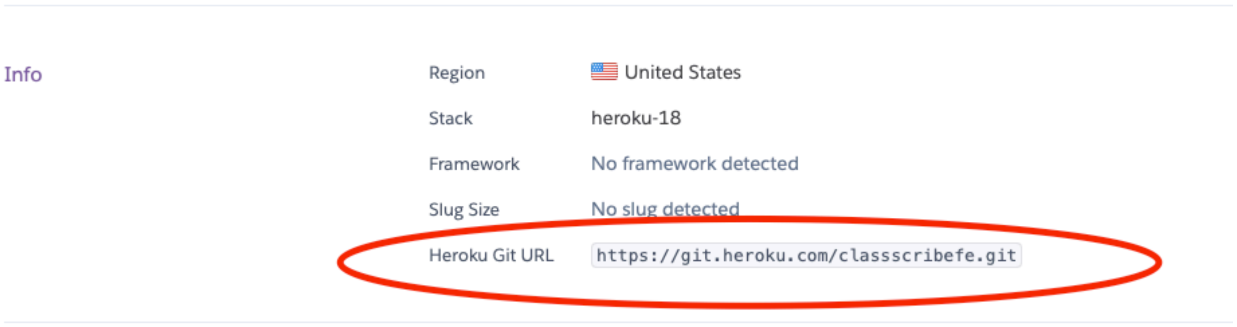
Custom domains allow you to access your app via one or more non-Heroku domain names (for example, `www.yourcustomdomain.com`)

8. Next, Enter these following commands into your terminal, which should be pointed to the root folder (Class-Scribe-master) of the git repo that you downloaded (you can use the cd command to navigate to the folder in terminal: type cd <the path to the folder>, where the

path is the path of the downloaded code and can be copied from the file explorer window you have opened):

a. **heroku login** (and follow on-screen instructions to login to your heroku account)

b. **git remote add heroku <INSERT_HEROKU_GIT_URL_FROM_HEROKU_BACKEND_APP_SETTINGS_PAGE> (1 line)**



Info		
	Region	United States
	Stack	heroku-18
	Framework	No framework detected
	Slug Size	No slug detected
	Heroku Git URL	`https://git.heroku.com/classscribefe.git`

c. **git subtree split --prefix src/classscribe master** (use results in step 5)

d. **heroku config:set DISABLE_COLLECTIONSTATIC=1**

e. **git push heroku<step c characters>:refs/heads/master**

f. **heroku run python manage.py migrate --run-syncdb**

. The backend should be up and running

9. Click on the frontend app (the other one of two Heroku apps you created) in your Heroku dashboard now:

a. go to the settings tab

b. select the nodejs buildpack under the buildpacks section

c. from the root folder (Class-Scribe-master), using a file explorer, go into src, then classscribe-fe, then go into the src folder and then open App.js with a text editor

- **d.** in the text editor, look for the line that starts with "export const base_url", and change the the text in the quotes on that line to the url of your BACKEND heroku app (the first app you made in these instructions, so for example, https://classscribe-be.herokuapp.com/), make sure to include the slash at the end of the url when you paste it.

10. Now in the terminal, pointed to the root of the git repo you downloaded:
    - **a.** switch the git heroku repo to the new git heroku repo of the frontend app by running the following commands
    - **b.** **git remote rm heroku**
    - **c.** **git remote add heroku <INSERT_HEROKU_GIT_URL_FROM_HEROKU_BACKEND_APP_SETTINGS_PAGE> (1 line)**
    - **d.** **git subtree split --prefix src/classscribe-fe master** (keep results for next step)
    - **e.** **git push heroku<step d results>:refs/heads/master**
    - **f.** Refresh the yourFrontEndAppName.herokuapp.com URL for the frontend should any errors occur initially when testing after successful deployment.

4. Results

The system solved all the problems it set out to solve, with all stakeholders of the project having had their functionality needs fulfilled. This fulfillment of needs has resulted in a backend, frontend, and lamp-end that work well together to perform all of our minimum and desired requirements. Students can use the lamps to link their notes to their accounts and be able to view their and others' notes, provided they opted to make their notebook publically viewable, as well as send notes or worksheets to their professors should this be a utility of our service their professor wishes to use.

Teachers can make an official notebook for each class using pages from students’ public notebooks, they can see attendance for a given class using the lamps, and they can label a public notebook as SDAC ready. University administrators can create course schedules, view the different courses that take place in a given classroom on a weekly basis, and search through existing course entries.

There is no meaningful way to have any real measurements for the primary function of our product as it adds digital capabilities and features to pen/paper notetaking, therefore not resulting in quantifiable “speed improvements” like a normal software project would do for a system it is supplanting. The only true metrics we would be able to gauge is doing a research study on long term productivity of students that utilize Class Scribe, which is beyond the scope of this class

5. Conclusions

The biggest takeaway from the research, development, and testing of this product has been the idea that a seismic shift is due in the way we take notes. It cannot be overstated how we are at a unique inflection point in our society where multiple companies are building technologies aimed towards finding solutions to problems in the education marketplace. Chief among them (and there are many, from textbooks and actual delivery of educational material from instructors), is the fragmentation across the methods students employ to take their notes. Solving this problem is paramount for higher success in education, and success in education creates success in society, this much is obvious.

Class Scribe, when executed properly, has broad implications and the ability to solve this problem. Addressing all the shortcomings of the methods before it, while adding unique and innovative spins of its own, Class Scribe is an impressive achievement that deserves interest and further development. Ideally, such a product would be deployed institution-wide, existing silently in the background and offering its users indispensable utility. Achieving that status is the task that a technology must undertake before being accepted as the de facto method for accomplishing a task, and Class Scribe is the only method of note-taking that exists today that is ideologically and technologically equipped to do so.

6. Future Work

There is sufficiently enough work left to be done for Class Scribe to mature into a serious and formidable contender going forward that it would be wise to carry the project forward to another research team in the future. While there are several tweaks and redesigns for the hardware components that would improve the experience of the product (such as higher quality components, custom chipsets tuned for computer vision algorithms, and a clip-on stand to allow the device to fit onto a fold-out school desk), these are beyond the scope of our research project. There also exist interesting research problems regarding human computer interaction, and studies to be completed on note-taking using the preliminary solution we have devised here with Class Scribe. However, these are similarly beyond the preset scope of this course for which Class Scribe was developed. Instead, there are many computer science-based challenges left to solve as well as innovative new software utilities to develop.

Further reducing human-computer interaction in the interest of a frictionless experience will require automatic detection of new pages being scanned. Removal of obstructions and generally unwanted objects in note scans will require advanced computer vision algorithms that can detect, remove, and replace data such as hands, blur, shadows, and pen/pencils. Authenticating a user currently requires a physical student ID card, an archaic methodology inevitably to be replaced by biometrics or some other methods such as handwriting verification. Annotation and digital inking tools are appreciative features that would make the web app a more suitable note-taking application comparable to OneNote or Evernote. Commenting and similar social features for notebooks, as well as a “looking for group” study group feature are welcome directions for the software to take. The possibilities are endless, and the opportunities Class Scribe presents are likewise.

References

Miller (2017, June 5). Apple Pencil: Improved 20ms latency, mark up support in iOS 11, new case w/ storage slot. Retrieved from https://9to5mac.com/2017/06/05/apple-pencil-improved-20ms-latency-mark-up-support-in-ios-11-new-case-w-storage-slot/

Mueller, P. A., & Oppenheimer, D. M. (2016). Technology and note-taking in the classroom, boardroom, hospital room, and courtroom. Trends in Neuroscience and Education, 5(3), 139–145. https://doi.org/10.1016/j.tine.2016.06.002