Adrian Rincón Jimenez and Ryan Johnson

Carroll College Classroom Analytics Manual

**CONTENTS**

[**INTRODUCTION** 1](#_Toc165370400)

[**USING THE APPLICATION** 2](#_Toc165370401)

[**Classroom Utilization Overview** 2](#_Toc165370402)

[**Classroom Usage Report** 5](#_Toc165370403)

[**Classroom Schedule** 8](#_Toc165370404)

[**Upload Data Page** 10](#_Toc165370405)

[**FOR IT / DEVELOPERS** 16](#_Toc165370406)

[**Requirements** 16](#_Toc165370407)

[**Setting Up the Application** 17](#_Toc165370410)

[**Changing the Database** 23](#_Toc165370419)

[**Changing HTML/JS/CSS** 23](#_Toc165370420)

[**Important Frontend Files** 24](#_Toc165370421)

[**Important Backend Files** 26](#_Toc165370425)

[**Model UML Diagrams / Development Gantt Chart** 30](#_Toc165370430)

[**Future Development** 31](#_Toc165370431)

**INTRODUCTION**

Welcome to the Carroll College Classroom Analytics scheduling software manual, designed to optimize your time management and enhance productivity. This manual is tailored to introduce you to our user-friendly scheduling tool, providing comprehensive guidance on its features, functionalities, and best practices for maximizing its benefits.

In this manual, you will discover how our analytics software empowers you to efficiently view, gather information and understand which classrooms are used at certain times throughout a semester.

Throughout this manual, we will walk you through the various aspects of our scheduling tool.

By the end of this manual, you will be equipped with the knowledge and skills to harness the full potential of our analytics software, enabling you to take control of your time, boost productivity, and understand how and when classrooms are being utilized.

# **USING THE APPLICATION**

## **Classroom Utilization Overview**

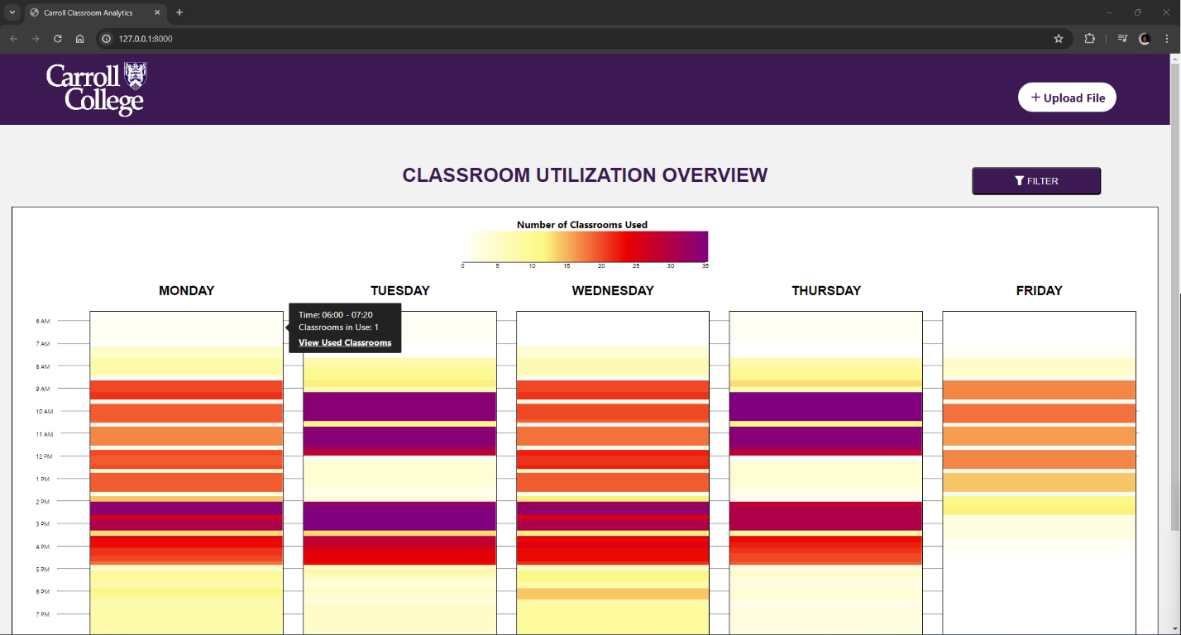
A screen shot of a classroom overview

Description automatically generatedThe heatmap view, as shown in Figure 1, is the home page. This heatmap is separated into columns by days of the week and into rows by time frames.

Figure 1: Heatmap Home Page

The colors displayed on the heatmap correspond to the number of classrooms that are being used during a specific block of time. Notice that a legend is provided directly above the schedule, explaining what different colors on the heatmap mean. Lighter colors, such as yellow and orange, indicate a lower number of classrooms are being used during that time frame. Conversely, darker colors, such as red and purple, indicate that more classrooms are being used at that time.­­­­­­

Figure 2: Heatmap Page Pop-up Example #1



The meaning of these colors is given further detail if a time block is hovered over with the mouse. A pop-up appears specifying the number of classrooms being utilized during a time period. For example, Figure 2 shows that on Mondays, from 6:00 AM – 7:00 AM, only one classroom is used, seen in the popup window. Meanwhile, Figure 3 demonstrates that 34 classrooms are being used on Thursdays from 11:00 AM – 11:50 AM. This pop-up also contains a link to a Classroom Usage Report, allowing the user to view a list of all classrooms being used during the time block in question. This report will be examined in more detail in the following section.

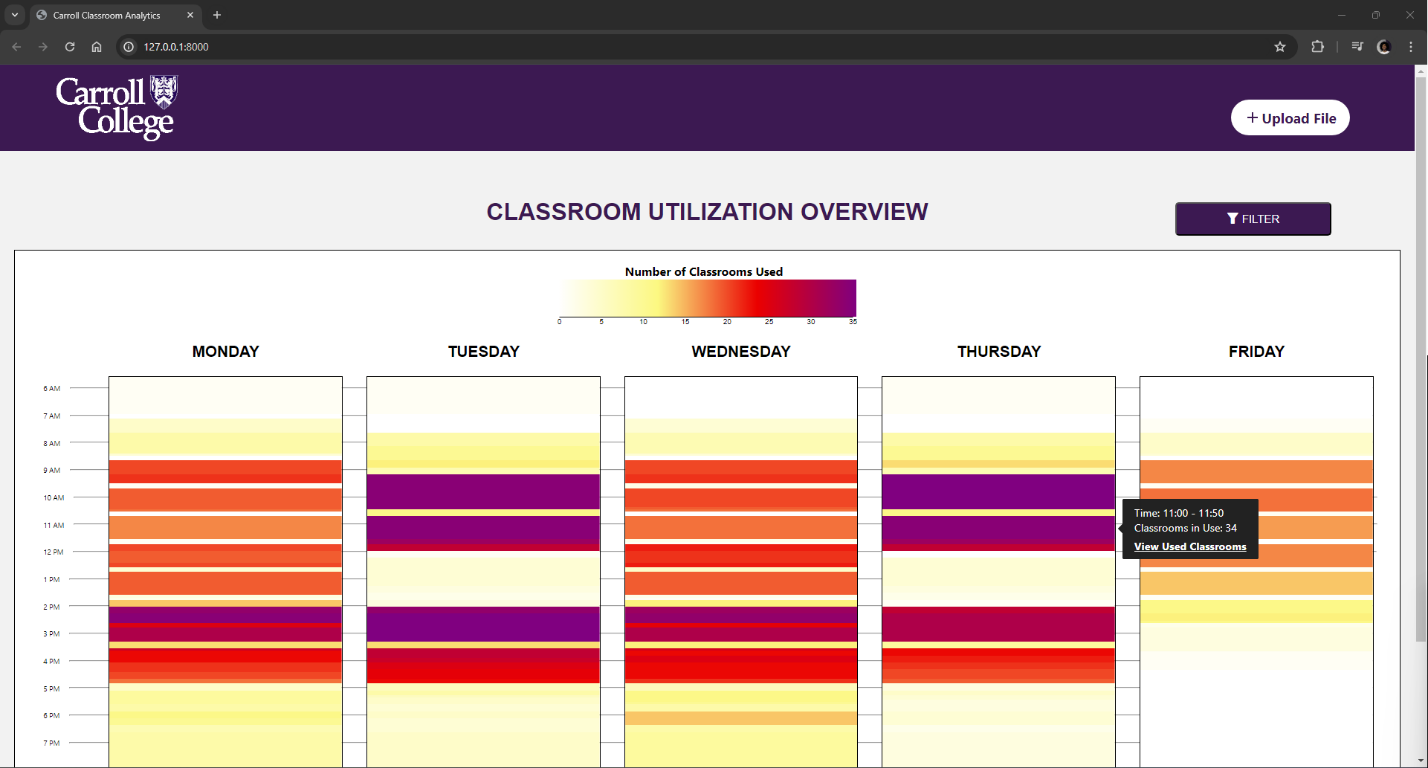


Figure 3: Heatmap Page Pop-up Example #2

A screen shot of a classroom overview

Description automatically generatedIn the upper-right corner, there lies a “Filter” button, used to toggle the display of the heat map between the various buildings on the Carroll College campus. Clicking on the button results in a dropdown list of each building hosting classrooms, shown in Figure 4.

Figure : Filter Dropdown List

A screen shot of a classroom overview

Description automatically generatedClicking on the checkbox to the left of any of these building names will change the heatmap display, showing the classroom distribution for the specified buildings only. For example, Figure 5 shows the appearance of the heatmap when the Civil Engineering building is selected. In this heatmap, only classrooms in the Civil Engineering building are being considered. Notice that the numbers in the legend have changed to reflect this change. More than one of these buildings may be selected to investigate the number of classrooms in several buildings together.

Figure 5: Selecting Civil Engineering from Filter Dropdown List

A screen shot of a classroom overview

Description automatically generatedClicking the “SELECT ALL” checkbox will do exactly as it suggests: all buildings in the list will be selected. However, if any buildings are already selected, hitting this box will instead do the direct opposite, deselecting all buildings. Lastly, notice the tan button in the upper-right corner of Figure 6: the “Upload File” button. Clicking this button will allow the user to upload a new Excel spreadsheet containing schedule data or classroom data.

Figure 6: Upload File Button

## **Classroom Usage Report**

Upon clicking a “View Used Classrooms” link from the heatmap view, a classroom usage report is shown as displayed in Figure 7. This view displays all classrooms being used within the selected time block, arranged in alphabetical order. For example, the following shows that SIMP-114 is being used on Mondays from 8:00 – 8:45 AM.

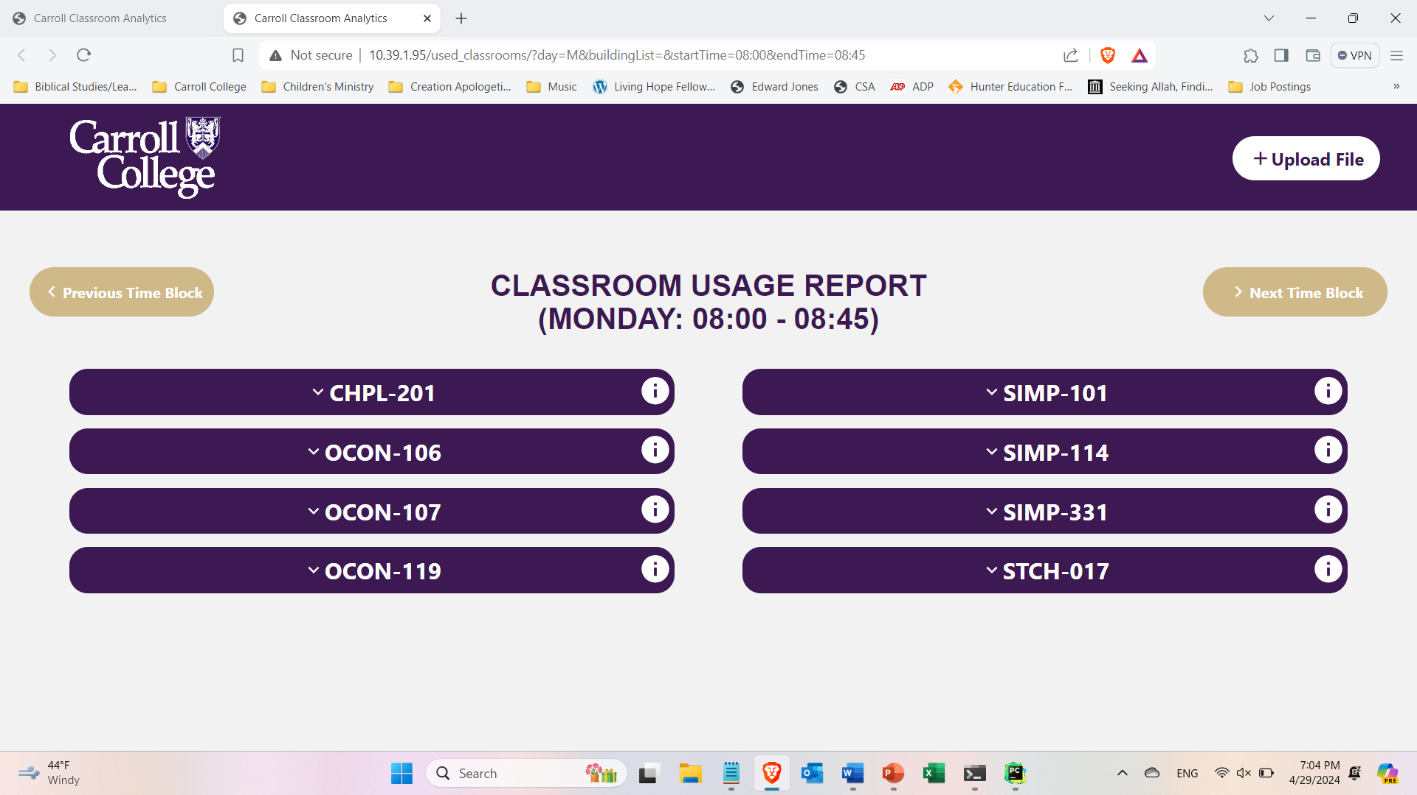


Figure : Sample Classroom Usage Report

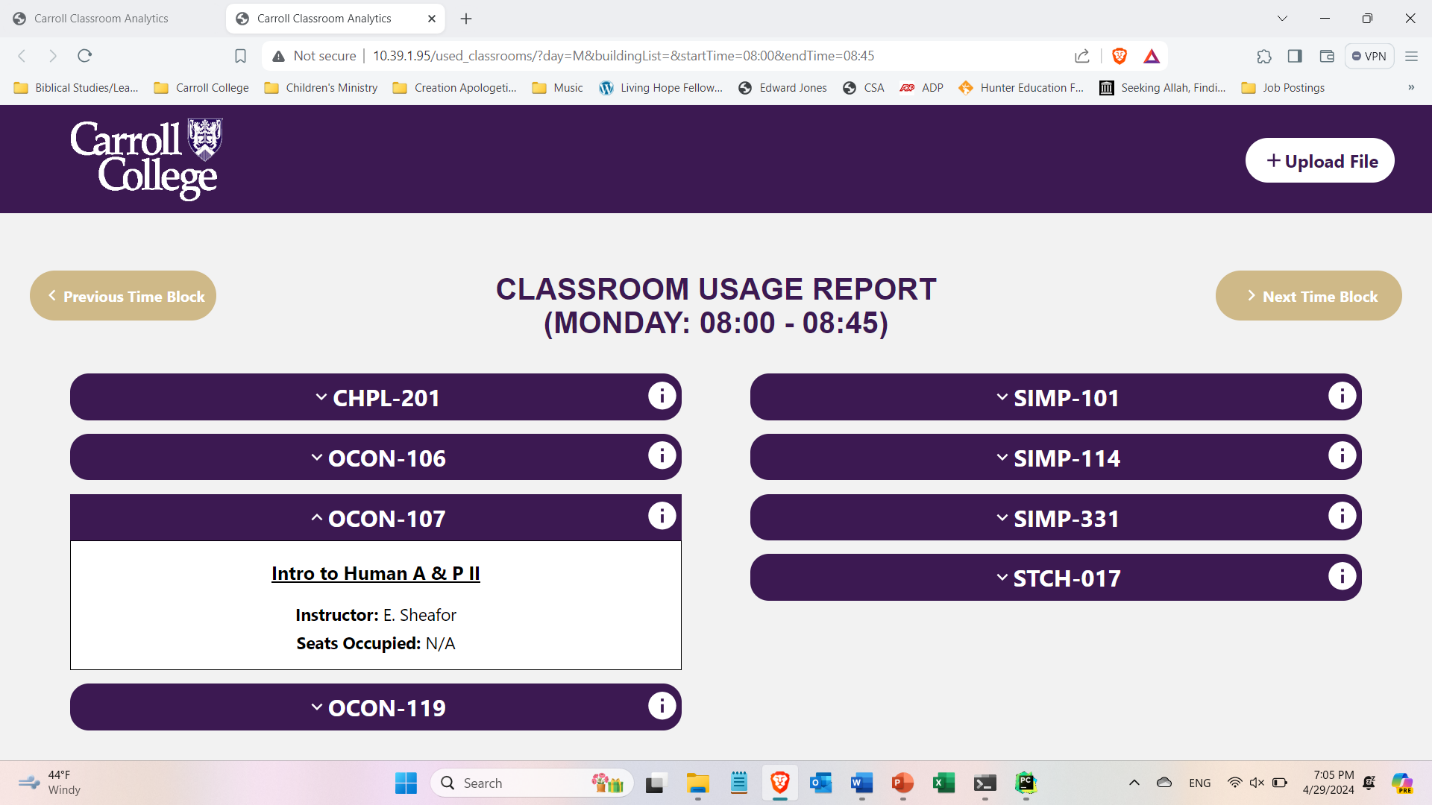
Note that the courses listed on this page are not necessarily constrained to the time period being displayed. Not all courses listed on this page run exactly from 8:00 – 845 AM. They may start earlier than 8:00 AM or may run later than 8:45 AM. This page solely states that the courses are being held for the entirety of the time period stated at the top of the screen.

Figure : Classroom Usage Report OCON-107 Example

Each of these classrooms is a dropdown. As such, clicking on any of the classrooms will display information about the courses being hosted in the classroom during the specified time period. In addition to the title of the course and its instructor, the dropdown shows the number of seats occupied during the course. This is ideal for discovering whether a classroom is a good size fit for the course it is currently hosting or whether the course should be moved to a different classroom.

Figure 8 shows the same screen as Figure 7 after clicking on OCON-107, showing that Intro to Human A & P II is held within OCON-107 on Mondays during 8:00 – 8:45 AM. Additionally, only half of the seats in the room are being used in this course, indicating that a smaller room could be a better fit for the course.

A screenshot of a computer

Description automatically generatedIf more than one course is listed as being in the same classroom during the same time period as another course, both will be listed in this dropdown. Figure 9 shows an example of this occurring in HAC-133. Both Drill & Conditioning 2 and 4 are listed as being in this classroom at the same time.

Figure : Classroom Usage Report Multiple Courses in a Classroom

There are also two buttons at the top of the page: “Previous Time Block” and “Next Time Block”, shown in Figure 10. These buttons allow the user to move between the immediately preceding/following time blocks for the currently selected day. For example, if the “Previous Time Block” were clicked on the example shown, the time period would shift to 7:30 – 8:00 AM. Similarly, the “Next Time Block” button would shift the time period to 8:45 – 8:50 AM, still on Monday. This feature allows a user to navigate throughout the day quicker than returning to the heatmap each time.

A screenshot of a computer

Description automatically generatedFigure : Classroom Usage Report Previous/Next Time Block Buttons

On the right-hand side, there is an information button correlated with each listed classroom, as shown in Figure 10. This button links to a schedule for the classroom, discussed in the following section.

## **Classroom Schedule**

A computer screen shot of a schedule

Description automatically generatedUpon the clicking of any information button on a classroom usage report. This schedule illustrates all times during which a classroom is being used to host a course throughout the week. This feature makes finding free time in a specific classroom quick and simple. An example classroom schedule can be seen below in Figure 11. This shows that OCON-107 holds classes from 8 AM – 12 PM on Fridays.

Figure : Classroom Schedule Page OCON-107 Example

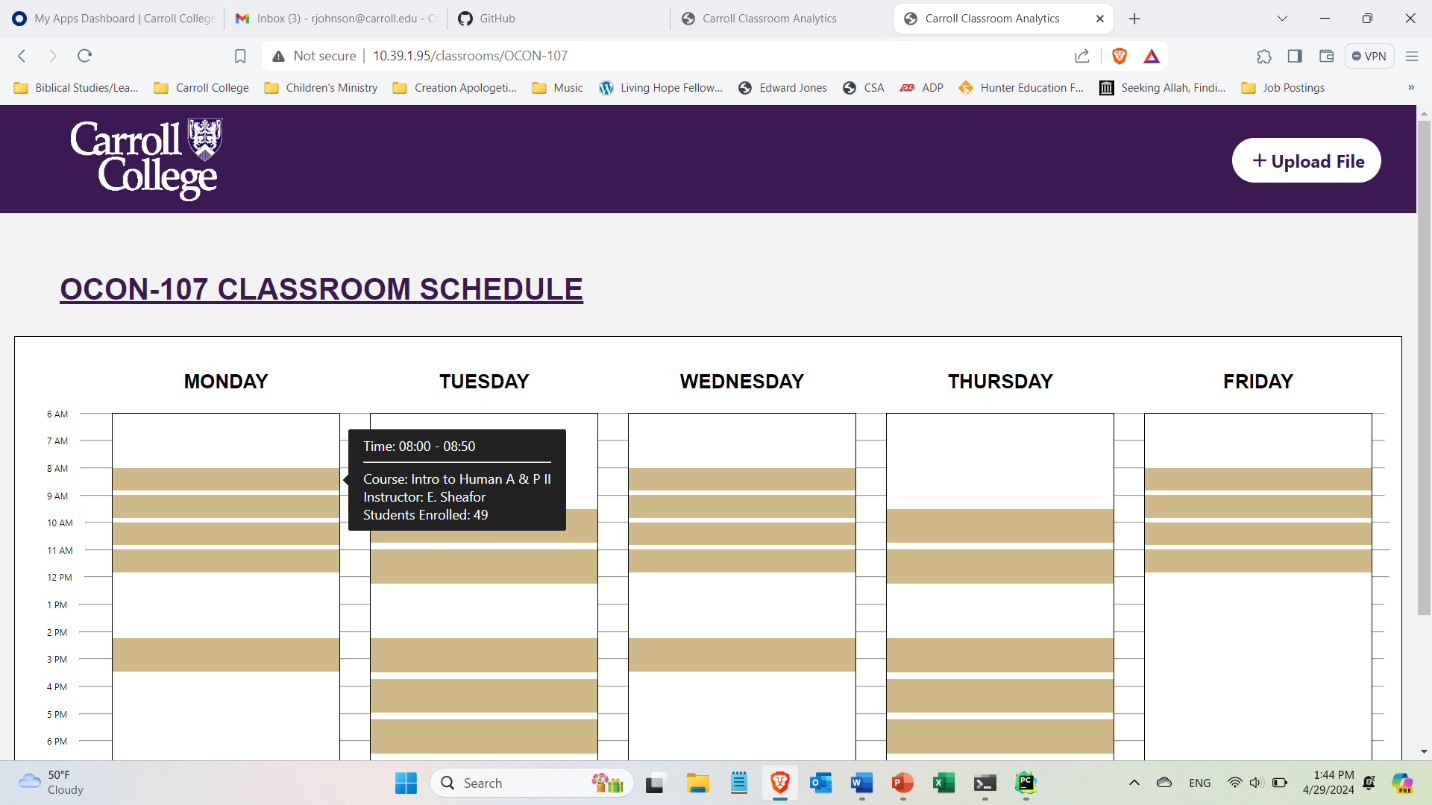
Hovering over any part of the classroom’s schedule prompts a pop-up that displays information about the course being held in the classroom during the time block. This is demonstrated in Figure 12, showing that Intro to Human A & P II uses OCON-107 from 8 – 8:50 AM.

Figure : OCON-107 Classroom Schedule Page Pop-up

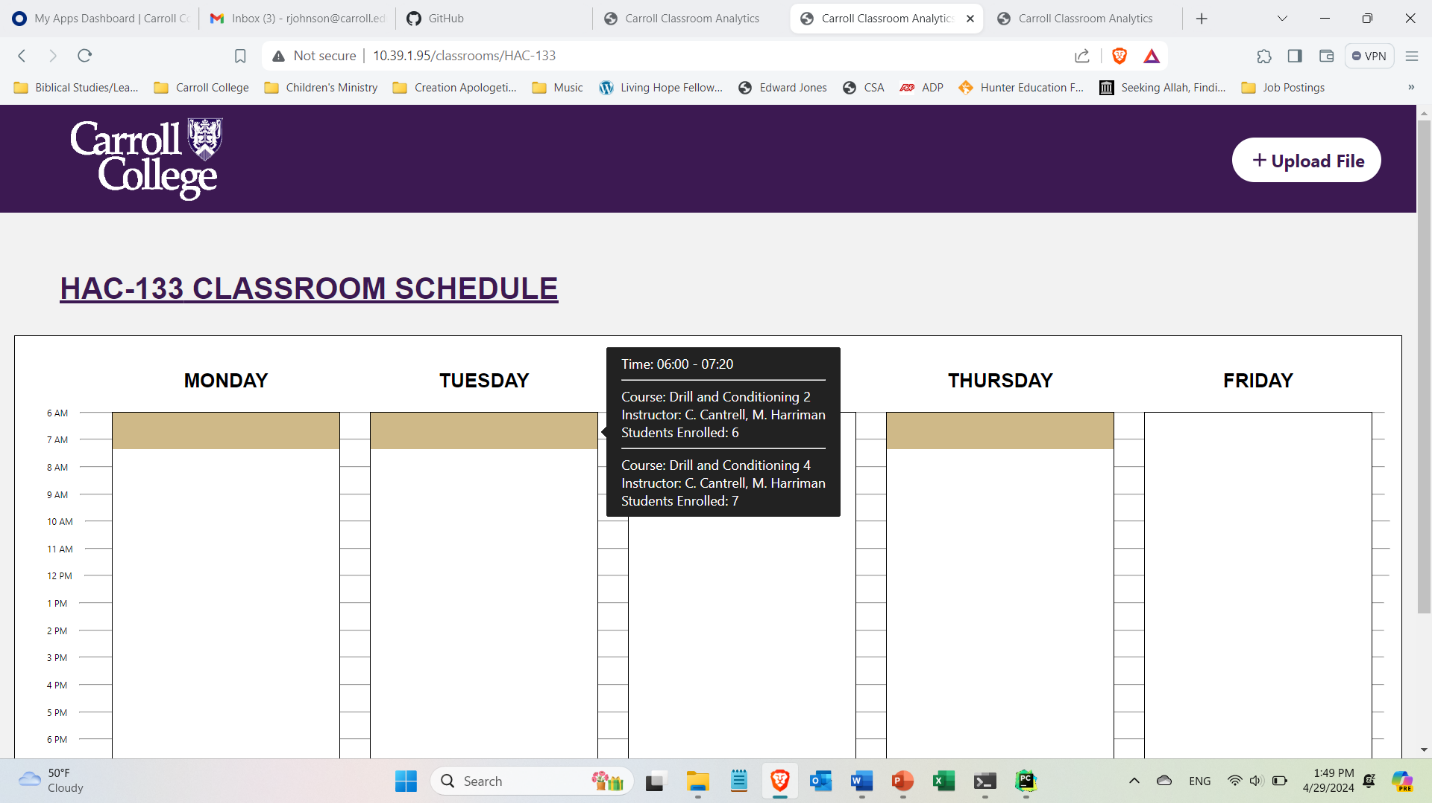
If there are several courses listed with the same classroom for a single time period, each course is listed in this pop-up, displaying similar details for each course. Figure 13 shows how several courses are listed in HAC-133 from 6:00 – 7:20 AM on Tuesdays. The information for each of these courses is displayed independently, with a line separating the information for each course.

Figure : Classroom Schedule Pop-up w/ several Courses in same Time Block

## **Upload Data Page**

A screenshot of a upload data

Description automatically generatedUpon clicking the “Upload File” button in the upper-right corner of the app, a user is taken to the page displayed in Figure 14. This page allows the user to upload new data into the application. This data must be uploaded as an Excel spreadsheet.

Figure : Upload Data Page

**Uploading Schedule Data**

There are two types of spreadsheets that may be uploaded, as indicated by the presence of two different upload buttons on this page. We will start by demonstrating the ability to upload a course schedule for a single semester. This schedule should include all courses offered during a semester, as well as any other data associated with the course, such as its classroom, start/end times, and number of students enrolled. To upload a schedule, begin by selecting the “Upload Schedule Data” button, as shown in Figure 15.

A screenshot of a upload data

Description automatically generatedNext, click on “Choose Schedule Data.” This opens a separate window, prompting one to select a file from your local machine. Note that only Excel files are able to be uploaded. Find the location of the desired file to upload. In Figure 16, Spring 2024 Schedule for Computer Science Senior

Figure : Upload Schedule Data Section

Project.xlsx is selected.Once the correct file is selected, click the “Open” button in the lower-right corner of the pop-up window to choose the file.

A screenshot of a computer

Description automatically generatedFigure : Opening Spring 2024 Schedule for Computer Science Senior Project.xlsx file

The name of the selected file should now appear directly beneath the “Choose Schedule Spreadsheet” button, as shown in Figure 17. The file to be uploaded may be changed at any time before starting the upload by clicking the “Choose Schedule Spreadsheet” button again. Once the name of the correct file is shown, click the “Upload File” button to begin the upload.

A screenshot of a computer

Description automatically generatedFigure : Showing “Upload File” button and file name under “Choose Schedule Spreadsheet” Button

A screenshot of a computer

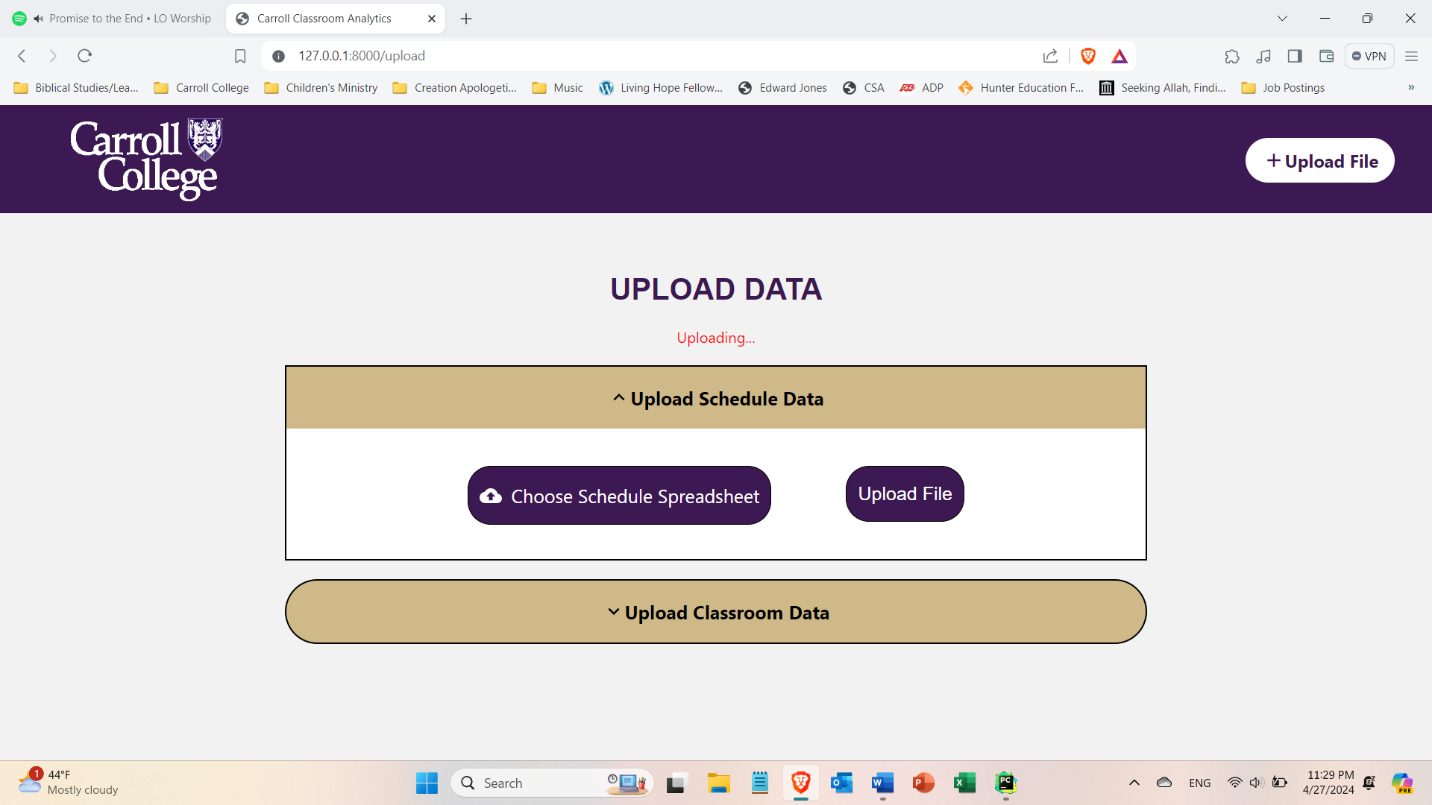
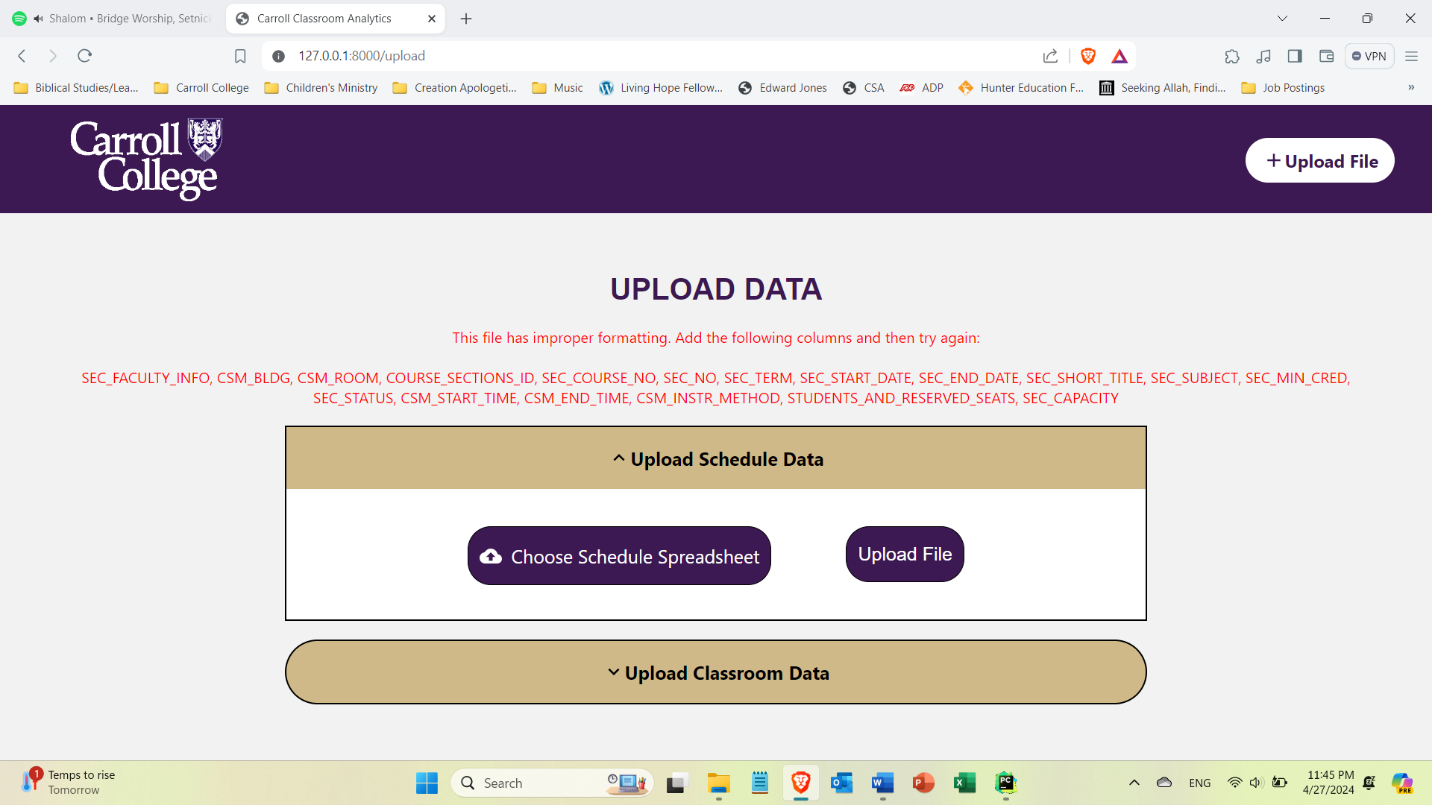
Description automatically generatedAssuming the selected file has no issues, a notification will appear, stating that the upload is in progress, as shown in Figure 18. Once the file has been successfully uploaded, a success message will be displayed, as shown in Figure 19. Note that depending upon the size of the file being uploaded, it could take up to 60 seconds for the upload to complete.

Figure : Uploading Notification

Figure : Upload Schedule Data Successful Upload

For a spreadsheet to be uploaded successfully, it must contain all the necessary data. As such, there are several required columns in all schedule spreadsheets. If a user attempts to upload a file that is missing any of these required columns, the upload will fail. An error message will be displayed, showing all the required columns that are missing from the selected file. Figure 20 shows a sample error message from submitting a spreadsheet not containing any of the required columns. Note that the file upload has failed. The name of each missing column is listed out in the error message.

Figure : Upload Schedule Data Error

It should be noted that schedule data is vital to the functionality of this application. Without schedule data, no heatmap will be displayed on the overall classroom utilization page, rendering the application useless. The homepage with no schedule data present is shown in Figure 21. As such, make sure to upload a semester course schedule before attempting to use any other functionality within the application.

A screenshot of a computer

Description automatically generatedFigure : Heatmap Display with No Uploaded Schedule Data

This application is currently only capable of displaying one semester’s worth of course data. Each time a new course schedule is uploaded, all schedule data previously uploaded will be REPLACED. Any schedule data previously available will be removed and replaced with the new data. As such, a new .xlsx file must be uploaded any time the user wishes to view data from a different semester.

*NOTE: In an effort to promote efficiency, all columns required for a course schedule spreadsheet upload have been listed below. Any .xlsx file to be uploaded as course schedule data must contain each of the following as the title of a column (order doesn’t matter):*

* *SEC\_TERM*
* *SEC\_FACULTY\_INFO*
* *CSM\_BLDG*
* *CSM\_ROOM*
* *COURSE\_SECTIONS\_ID*
* *SEC\_COURSE\_NO*
* *SEC\_NO*
* *SEC\_TERM*
* *SEC\_START\_DATE*
* *SEC\_END\_DATE*
* *SEC\_SHORT\_TITLE*
* *SEC\_SUBJECT*
* *SEC\_MIN\_CRED*
* *SEC\_STATUS*
* *CSM\_START\_TIME*
* *CSM\_END\_TIME*
* *CSM\_INSTR\_METHOD*
* *STUDENTS\_AND\_RESERVED\_SEATS*
* *SEC\_CAPACITY*

**Uploading Classroom Data**

In addition to uploading semester course schedule data, classroom data may also be uploaded. This classroom data is what provides data about specific classrooms, such as the number of seats present in the room. Without this data, no information will be available about how many seats are available in a classroom when hosting a given course (as displayed on a Classroom Usage Report).

Start the upload process by clicking the “Upload Classroom Data” button to show the dropdown, as shown in Figure 22. From this point, classroom data may be uploaded in a manner very similar to uploading a course schedule. As such, the steps provided earlier will not be repeated here. To upload classroom data, follow the same steps provided in the preceding section. Similar to schedule data, uploading classroom data replaces any previously uploaded classroom data. As such, whenever a classroom spreadsheet in uploaded, ensure that all desired classrooms are listed in the spreadsheet.

A screenshot of a computer

Description automatically generatedFigure : Upload Classroom Data Section

*NOTE: Any .xlsx file to be uploaded as classroom data must contain each of the following as the title of a column (order doesn’t matter):*

* *Building Information*
* *Room Number*
* *Number of Student Seats in Room*
* *Width of Room*
* *Length of Room*
* *Number of Projectors in Room*
* *Does room have any of the following?*
* *Any other things of note in Room (TV or Periodic Table poster)*
* *Notes*

# **FOR IT / DEVELOPERS**

## **Requirements**

### **User Requirements**

**Hardware**

* 1 GB of RAM
* 1.5 GB Hard drive

**Web browser/Internet**

You will need a modern web browser and an Internet connection to run this application. We utilized Google Chrome.

### **Deployment Requirements**

**Hardware**

* 4GB of RAM
* 10 GB storage space

**Operating System**

Windows 10/11 and macOS 10.10 have been tested to work with this application.

**Software**

**Backend**

* Python 3.12
  + Django Ver. 5.0.1
* Mysql Ver. 8.0.34

**Frontend**

* Node.js Ver. 20.11.0
  + npm Ver. 0.1.1

Other packages and version requirements are stated in the *requirements.txt* file.

**Web browser/Internet**

You will need a modern web browser of any kind. We utilized Google Chrome. Internet connection is also needed to run the web application.

## **Setting Up the Application**

### **Creating the Database**

1. Install MySQL 8.0.34 on your local machine(can be downloaded at [dev.mysql.com/downloads/mysql/](https://dev.mysql.com/downloads/mysql/)). Ensure that it is configured correctly by typing ***mysql --version***. If MySQL is installed correctly, this command should output something similar to ***mysql Ver 8.0.34.***
2. Log into the root account: ***sudo mysql -u root***
3. Once logged into the root account, create a new database:

***CREATE DATABASE your\_database\_name;***

In place of *your\_database\_name*, type the name you would like to give your database. **TAKE NOTE OF THIS NAME.** You’ll need it again later.

1. Create a new user:

***CREATE USER 'your\_user\_name'@'localhost' IDENTIFIED BY 'your\_user\_password;******CREATE USER 'caroll\_user'@'%' IDENTIFIED BY 'your\_user\_password';***

In place of *your\_user\_name* and *your\_user\_password*, type the username and password you would like to give your new user. Again, **TAKE NOTE OF THE USERNAME AND PASSWORD.** You’ll need both of these later in the setup process.

1. Give this user privileges on the newly created database:

***GRANT ALL PRIVILEGES ON your\_database\_name.\* TO 'your\_user\_name'@'localhost' WITH GRANT OPTION;***

***GRANT ALL PRIVILEGES ON your\_database\_name.\* TO 'your\_user\_name'@'%' WITH GRANT OPTION;***

Make sure to use the database name and user name you used in Steps 2 & 3.

1. Log out of the root account: ***quit***
2. Log in as this newly created user, using the same username and database name as used in Steps 3-4. You will be prompted for a password – use the same password created in Step 4.

***mysql -u your\_user\_name -p your\_database\_name***

1. If you can successfully log in, your database is all set up! Log out of this user before continuing.

### **Installing Dependencies**

1. Install Git on your machine (can be downloaded at [git-scm.com/downloads](https://git-scm.com/downloads)). Ensure you have Git installed on your machine by typing ***git --version***. If installed correctly, you should receive output similar to ***git version 2.41.***
2. Install Node.js on your machine (can be downloaded at [nodejs.org/en/download](https://nodejs.org/en/download)).
3. Install npm on your machine (can be downloaded at [www.npmjs.com/package/download](http://www.npmjs.com/package/download)). Ensure you have Git installed on your machine by typing ***npm --version***. If installed correctly, you should receive output similar to ***10.0.0.***
4. Install Python 3.12 on your machine (can be downloaded at [python.org/downloads/](https://www.python.org/downloads/)). Ensure you have Python installed on your machine by typing ***python3 --version*** if on Linux/MacOS or ***python --version*** if on Windows. If installed correctly, you should receive output similar to ***Python 3.12.***
5. Ensure that you have the most recent version of pip installed. If using macOS or Linux, use this command: ***sudo pip3 install -U pip***. If using Windows, use: ***pip install -U pip***.
6. Install Django: ***pip3 install django***
7. Clone the repository from GitHub:

***git clone*** [***https://github.com/rjohnson05/CarrollClassroomAnalytics***](https://github.com/rjohnson05/CarrollClassroomAnalytics)

1. Move into the parent directory of the project: ***cd CarrollClassroomAnalytics***
2. Install necessary back end dependencies: ***pip3 install -r requirements.txt***
3. Move into the frontend directory: ***cd frontend***
4. Install frontend dependencies: ***npm install***
5. Move back to the parent directory of the project: ***cd ..***

### **Setting Environment Variables**

1. Ensure that you are in the parent directory. Make a copy of the *.env.template* file and rename it to *.env*. In Linux, this can be done with this command: ***cp .env.template .env***. In Windows, this can be done with the following command: ***copy .env.template .env***.
2. Generate a secret key: ***openssl rand -base64 32***. Copy the output of this command.
3. Paste your generated secret key into the *.env* file after *SECRET\_KEY=*
4. Fill in the database name, username and user password used when creating the database.

### **Creating the Database Structure**

1. Ensure that you are in the parent directory. First create the migration files needed to load the database structure: ***python3 manage.py makemigrations api***
2. Load the database structure: ***python3 manage.py migrate***

### **Testing the Application**

Testing the application runs all unit tests within the *api/tests/test\_services.py* file. All tests for the methods within *api/services.py* can be run to ensure the application’s backend logic is behaving correctly. These tests are completely optional. If you don’t want to test the application, skip to the next section (“Building Static Files”) to continue setting the app up.

1. Log into the root MySQL account: ***sudo mysql -u root***
2. Give the user created earlier privileges on the testing database

***GRANT ALL PRIVILEGES ON test\_`your\_database\_name`.\* TO 'your\_user\_name'@'localhost' WITH GRANT OPTION;***

***GRANT ALL PRIVILEGES ON test\_`your\_database\_name`.\* TO 'your\_user\_name'@'%' WITH GRANT OPTION;***

Notice that the name of the testing database is the same as that of the production database, but with *test\_* placed in front. So, if the name of the production database is *database*, the name of the testing database would be *test\_database*. Make sure to use the same user as created earlier.

1. Log out of the root account: ***quit***
2. Ensure that you are in the root directory before running the tests: ***python3 manage.py test***

### **Building Static Files**

1. Ensure that you are in the parent directory. Move to the frontend directory: ***cd frontend***
2. Build the front end static files for production: ***npm run build***
3. Move back to the parent directory: ***cd ..***
4. Build static files in the back end: ***python3 manage.py collectstatic***
5. There should now be a *static/* directory within the parent directory.

### **Setting Up a WSGI Server & Web Server**

To set up the application for production, both a WSGI server and web server need to be set up. If using Linux, this can be done using Gunicorn and Nginx, as shown below. For alternate setups, see Dunicorn’s deployment documentation: <https://docs.gunicorn.org/en/latest/deploy.html>

1. Ensure that you are in the parent directory, and then move the static files to */var/www/*:

***sudo cp -r static /var/www***

1. Create a Gunicorn socket: ***sudo nano /etc/systemd/system/gunicorn.socket***
2. Inside *gunicorn.socket*, place the following:

***[Unit]***

***Description=gunicorn socket***

***[Socket]***

***ListenStream=/run/gunicorn.sock***

***[Install]***

***WantedBy=sockets.target***

1. Create a Gunicorn service: ***sudo nano /etc/systemd/system/gunicorn.service***
2. Inside *gunicorn.service*, place the following:

***[Unit]***

***Description=gunicorn daemon***

***Requires=gunicorn.socket***

***After=network.target***

***[Service]***

***User=student***

***Group=www-data***

***WorkingDirectory=/home/student/CarrollClassroomAnalytics***

***ExecStart=/home/student/.local/bin/gunicorn \***

***--access-logfile - \***

***--workers 3 \***

***--bind unix:/run/gunicorn.sock \***

***carroll\_classroom\_analytics.wsgi:application***

***[Install]***

***WantedBy=multi-user.target***

1. Start the socket: ***sudo systemctl start gunicorn.socket***
2. Enable the socket: ***sudo systemctl enable gunicorn.socket***
3. Check the status of the socket: ***sudo systemctl status gunicorn.socket.***

You should see that the gunicorn socket is active and enabled.

1. Test the gunicorn service by sending a testing request to the socket: ***curl –unix-socket /run/gunicorn.sock localhost***. You should receive an HTML response if it is working correctly.
2. Install Nginx on your machine (can be downloaded at <https://nginx.org/en/download.html>)
3. Create a new Nginx server block:

***sudo nano /etc/nginx/sites-available/carroll\_classrooms\_analytics.conf***

1. In *carroll\_classrooms\_analytics.conf*, type the following:

***server {***

***listen 80;***

***server\_name 10.39.1.95;***

***location /static/ {***

***root /var/www;***

***}***

***location / {***

***include proxy\_params;***

***proxy\_pass http://unix:/run/gunicorn.sock;***

***}***

***}***

1. Link this newly created file to the *sited-enabled* directory:

***sudo ln -s /etc/nginx/sites-available/carroll\_classrooms\_analytics.conf /etc/nginx/sites-enabled***

1. Restart Nginx: ***sudo systemctl restart nginx***

### **Launching the Application**

1. Launch the program by running this command: ***python3 manage.py runserver***
2. The application should now be running! Test this by entering the IP address of the machine into a web browser on another machine. You should now be able to see the home page of the application, titled “Classroom Utilization Overview”.

## **Changing the Database**

To change the database, whether it be editing the name of an attribute, adding new attributes, or deleting an attribute, you will need to edit the ***api/models.py*** file. This file contains all of the models for the application, including the Classroom, Course, and Instructor models. Each of these models creates a separate table in the database. Within each class, the model contains attributes (ID, name, start time, etc.), representing the columns created in the database.

These models may be changed, as necessary. To add an attribute, add a new line at the end of the model and follow the existing syntax with the new column information. However, there are a couple things to keep in mind when altering the structure of these models:

* The classroom model is linked to the course model by a foreign key. This foreign key should not be removed.
* If a new attribute is added to a model, the value of this new column will have to be specified for any data already present within the database that was added before the new column existed.
* If the names of current attributes are adjusted, these names will also need to be adjusted wherever used in the ***api/services.py*** file (where the database is queried).

Once the desired changes have been made to ***api/models.py***, these changes need to be reflected in the database. To do so:

1. Ensure you are in the parent directory of the project
2. Generate migration files: ***python3 manage.py makemigrations***
3. Apply these changes to the database: ***python3 manage.py migrate***

## **Changing HTML/JS/CSS**

All JS and CSS components of the application are found within the ***frontend/src/*** directory.

Each type of file is found within these directories:

* .js: ***frontend/src/components***
* .css: ***frontend/src/css***
* .png: ***frontend/src/images***

The home page of the application is *HeatmapSchedule.js*. This file contains the code for rendering the Overall Classroom Utilization page. This home page is loaded in place of a 404 page. In the case of a 404 error, the request is redirected to display this page instead of an error.

## **Important Frontend Files**

The following files and code examples abridges our most important files to the frontend of our application.

### **HeatmapSchedule.js**

This .js file displays the main page, showing a heat map that contains the number of used classrooms during each time block. Lighter colors, such as yellow and orange indicate less classroom usage, while darker colors such as purple indicate more classroom usage. The ‘return’ statement at the end is where the display is rendered. The *useEffect()* function below fetches the data detailing the number of classes per time block from the database upon startup, then reloads whenever the building filter is change. This function also loads the names of the buildings from the database in the buildings filter.

useEffect(() => {  
async function fetchData() {  
 await loadData();  
 if (firstRender) {  
 await loadFilter();  
 }  
 }  
  
 fetchData();  
}, [buildingFilter]);

### **ClassroomDaySchedule.js**

This file is a component to display the course being used during a specific time block during the day. This differs from the heatmap component from the main page as this displays only one color used to show when the classroom contains a course. When a time block is hovered over, a tooltip shows information about the course using the classroom during the time block. The *calculateMinutes()* constant calculates the start time and end time for the blocks. This is used for forming the size of the colored and white blocks on the day calendar.

const calculateMinutes = (startTime, endTime) => {  
 startTime = startTime.split(":");  
 endTime = endTime.split(":");  
 const minutes = endTime[1] - startTime[1];  
 // Accounts for the noonday hour change from 12 to 1  
 if ((parseInt(startTime[0]) > 1 && endTime[0] === '1')) {  
 return (60 + minutes) / (1.6);  
 }  
 // Handles all other hour changes  
 const hourMinutes = 60 \* (endTime[0] - startTime[0]);  
 return (hourMinutes + minutes) / 5;  
}

### **FileUpload.js**

This file displays the page for uploading Excel spreadsheets. This data replaces any data currently in the database. The *handleFileChange()* constant changes the file being uploaded whenever a file is selected using one of the file browsers. This file information will be sent to the database if the user submits before choosing another file.

const handleFileChange = (e) => {  
 if (e.target.id === "classroom-chooser") {  
 setClassroomFile(e.target.files[0]);  
 } else {  
 setScheduleFile(e.target.files[0]);  
 }  
}

## **Important Backend Files**

The following summarizes the functionality of the four most important files to the backend of this application.

### **models.py**

This file contains all of the models for the application, setting the database schema. It holds the Classroom model and the Course model. The following shows the Classroom model. Note than if any new buildings are added to campus, they must be added to the *BUILDINGS* dictionary located here.

class Classroom(models.Model):BUILDINGS = {  
 "": "",  
 "CENG": "Civil Engineering",  
 "CHPL": "All Saints Chapel",  
 "CUBE": "Cube",  
 "EQCT": "Equine Center",  
 "FSCT": "Fortin Science Center Labs",  
 "GUAD": "Guadalupe Hall",  
 "HAC": "Hunthausen Activity Center",  
 "OCON": "O'Connell Hall",  
 "OFCP": "Off-Campus",  
 "PCCC": "Perkins Call Canine Center",  
 "PECT": "PE Center",  
 "SIMP": "Simperman Hall",  
 "STCH": "St. Charles Hall",  
 "WBAR": "Waterbarn",  
 }  
  
 id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 building = models.CharField(max\_length=255, choices=BUILDINGS)  
 room\_num = models.CharField(max\_length=255)  
  
 occupancy = models.FloatField(default=None, blank=True, null=True)  
 width = models.IntegerField(default=None, blank=True, null=True)  
 length = models.IntegerField(default=None, blank=True, null=True)  
 projector\_num = models.IntegerField(default=None, blank=True, null=True)  
 features = models.CharField(max\_length=255, default=None, blank=True, null=True)  
 notes = models.CharField(max\_length=255, default=None, blank=True, null=True)  
  
 def \_\_str\_\_(self):  
 return self.name

### **services.py**

This file contains all of the business logic for the back end of the application, such as calculating time blocks, determining the number of classes within a time period, and finding all times a classroom is used. These methods are called from *views.py,* which sends the calculated information to the front end. Here is a snippet of the *upload\_schedule\_data()* method, found within this file.

def upload\_schedule\_data(file): # Delete all data to prevent different semesters being present in the DB  
 Course.objects.all().delete()  
  
 # Create new courses from the uploaded file  
 for index, row in df.iterrows():  
 instructor, \_ = Instructor.objects.get\_or\_create(  
 name=row['SEC\_FACULTY\_INFO'],  
 )  
 logger.debug(f"Instructor {instructor} present")  
  
 classroom = None  
 # Create a classroom object if it doesn't already exist  
 if not pd.isna(row['CSM\_BLDG']):  
 classroom, created = Classroom.objects.get\_or\_create(  
 name=row['CSM\_BLDG'] + "-" + str(row['CSM\_ROOM']),  
 building=row['CSM\_BLDG'],  
 room\_num=row['CSM\_ROOM'],  
 )  
 if created:  
 logger.debug(f"Classroom {classroom} created")  
  
 # Create the course object  
 day\_string = calculate\_day\_string(row)  
 course = Course.objects.create(  
 section\_id=None if pd.isna(row['COURSE\_SECTIONS\_ID']) else row['COURSE\_SECTIONS\_ID'],  
 course\_num=None if pd.isna(row['SEC\_COURSE\_NO']) else row['SEC\_COURSE\_NO'],  
 section\_num=None if pd.isna(row['SEC\_NO']) else row['SEC\_NO'],  
 term=None if pd.isna(row['SEC\_TERM']) else row['SEC\_TERM'],  
 start\_date=datetime.strptime(row['SEC\_START\_DATE'], '%b %d %Y').date(),  
 end\_date=datetime.strptime(row['SEC\_END\_DATE'], '%b %d %Y').date(),  
 name=row['SEC\_SHORT\_TITLE'],  
 subject=row['SEC\_SUBJECT'],  
 min\_credits=None if pd.isna(row['SEC\_MIN\_CRED']) else row['SEC\_MIN\_CRED'],  
 status=None if pd.isna(row['SEC\_STATUS']) else row['SEC\_STATUS'],  
 start\_time=None if pd.isna(row['CSM\_START\_TIME']) else datetime.strptime(row['CSM\_START\_TIME'],  
 '%I:%M%p').time(),  
 end\_time=None if pd.isna(row['CSM\_END\_TIME']) else datetime.strptime(row['CSM\_END\_TIME'],  
 '%I:%M%p').time(),  
 day=day\_string,  
 classroom=classroom if classroom is not None else None,  
 instruction\_method=row['CSM\_INSTR\_METHOD'],  
 instructor=instructor,  
 enrolled=None if pd.isna(row['STUDENTS\_AND\_RESERVED\_SEATS']) else row['STUDENTS\_AND\_RESERVED\_SEATS'],  
 capacity=None if pd.isna(row['SEC\_CAPACITY']) else row['SEC\_CAPACITY'],  
 )  
 logger.debug(f"Course {course} created")  
  
 logger.info(f"New Course Schedule Spreadsheet Uploaded: {file.name}")  
 return True, None

This method is responsible for creating classroom and course objects from an uploaded schedule Excel spreadsheet and populating the database with these objects.

### **urls.py**

This file contains a list of possible URL paths for the application, mapping each URL to a method in *views.py*. When HTTP requests are received, the URL is first sent to *carroll\_classroom\_analytics/views.py*. If the URL does not contain *api/*, it is passed to the front end router for displaying the correct page of the application. However, if the URL contains *api/*, the request is passed to this file, where the request is compared against each of the URL patterns shown below (first parameter for each path) and passed to the correct method in *api/views.py* (second parameter for each path). For example, a URL of *localhost:8000/get\_number\_classes/* matches the second pattern in this list, so being passed to the *get\_numer\_classes()* method within *api/views.py*. The third parameter, *name*, gives each URL pattern a unique identifier for when a redirect is necessary. This requires the URL pattern to be changed in this location only, rather than every time a redirect is made.

urlpatterns = [  
 path('get\_number\_classes/', views.get\_number\_classes, name="get\_number\_classes"),  
 path('get\_building\_names/', views.get\_building\_names, name="get\_building\_names"),  
 path('get\_used\_classrooms/', views.get\_used\_classrooms, name="get\_used\_classrooms"),  
 path('get\_past\_time/', views.get\_past\_time, name="get\_past\_time"),  
 path('get\_next\_time/', views.get\_next\_time, name="get\_next\_time"),  
 path('get\_classroom\_data/', views.get\_classroom\_data, name="get\_classroom\_data"),  
 path('upload\_file/', views.upload\_file, name="upload\_file"),  
]

### **views.py**

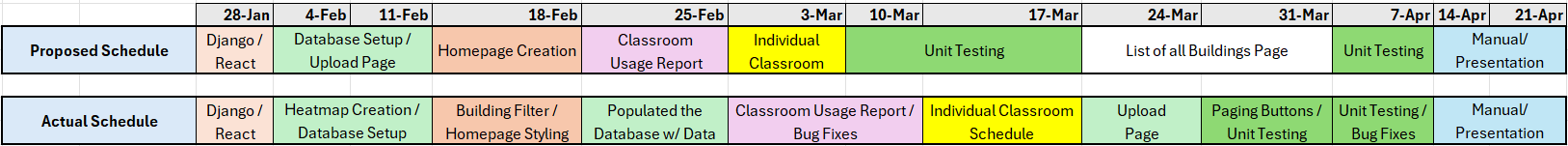
This file contains all the methods responding to HTTP requests, passed from *urls.py*. The following method from *views.py* is responsible for returning the weekly schedule for a single classroom. It returns a single list containing two dictionaries: the first storing the time blocks and the second storing data about the courses that are held in the classroom.

@api\_view(["GET"])  
def get\_classroom\_data(request: Request) -> Response:classroom\_name = request.GET.get("classroom")  
 courses = services.get\_classroom\_courses(classroom\_name)  
 logger.debug(f"get\_classroom\_data - Found courses held in

{classroom\_name}: {courses}")  
 return Response(courses)

## **Model UML Diagrams / Development Gantt Chart**

A screenshot of a computer program

Description automatically generated****

## **Future Development**

There are several features that could make this application more useful than its current form. Each of the following features serve as inspiration for future developers.

**Ability to View Unused Classrooms**

This application at present allows a user to view which classrooms are being utilized during a particular period of time. This functionality is detailed in the section titled “Classroom Usage Report.” However, what if the user has a desire to find *unused* classrooms during the same time period? Currently, the may be found through determining which classrooms are not listed. Yet, this task would be made easier if the user had the ability to toggle whether they wanted to see a list of used or unused classrooms.

**Classroom Specifications**

To promote efficiency, it would be helpful to provide details about each classroom on what is now the “Classroom Schedule” page. These details could include specs such as the number of seats, number of projectors, and size of the room. Having all classroom information inside the application would prevent having to switch between several different apps. Additionally, having the ability to navigate directly to a desired classroom to view its specifications, rather than having to go through the heatmap on the homepage would improve the user experience.

**Spreadsheet Editor**

In addition to importing semester scheduling data from a spreadsheet, the user may have a desire to manipulate the spreadsheet and export it. Adding a spreadsheet editor would allow the user to reduce the complexity of the master spreadsheet, making it easier to find specific information within the data as edits are made to the schedule.

**Genetic Algorithm Continuation**

The use of genetic algorithms can be used to provide a feasible semester course schedule. One of these algorithms was implemented during the Spring 2024 semester. Although this algorithm made many assumptions that make it impractical to use in the real world, additional development could enable this algorithm to create an initial course schedule, taking into consideration various constraints (professor time availability, time required for differing credit hours, etc.). Once an algorithm producing feasible solutions has been developed, it could be integrated into this application. The current algorithm may be found at [github.com/rjohnson05/CarrollClassroomOptimization](https://github.com/rjohnson05/CarrollClassroomOptimization).