
Semester.ly Documentation

Release 2.0

Semester.ly Technologies, LLC

Sep 28, 2022

CONTENTS

1 What We Believe In	3
1.1 Course registration should be easy	3
1.2 Education should be collaborative	3
1.3 Students know best	3
Python Module Index	83
Index	85

Note: Want your school on Semester.ly? We want it too! Want to see a new feature to help your peers? Let's make it happen. We want to help you make the impact you want to see. We'll even find you something impactful to work on if you're not sure where to start.

Built for students by students. Semester.ly is an open source web platform created to bring modern technology to the most difficult and archaic parts of higher education. It all started with one problem, universal across all college campuses: course registration is a pain. Spreadsheets, sticky notes, PDFs of course evaluations, and an outdated registration platform...it is all too much in the heat of classes and exams. We set out with the mission to make college more collaborative and more stress free.

WHAT WE BELIEVE IN

Today, we work to solve many more exciting problems in this space across many more universities. However, our fundamental beliefs remain the same:

1.1 Course registration should be easy

Picking the right classes should be quick and painless. We believe high quality, centralized, and shareable information makes for better decision making. By doing the legwork for you, Semester.ly gives you more time to study for your courses, and decreases the time spent studying which classes to take.

1.2 Education should be collaborative

Studies show the positive impact that friendship has in higher education classrooms. Having courses with friends and a tighter knit university community increases student success and retention. That's why Semester.ly helps students find courses with friends and helps new students make new friends in their classes.

1.3 Students know best

Universities can't keep up with technology. Most university systems aren't even mobile responsive! Forget about using social media. That's why Semester.ly is built by students, and always will be.

1.3.1 Installation

This guide will bring you through the steps of creating a local Semester.ly server and development environment. It will walk through the setup of the core ecosystems we work within: Django/Python and React/Node/JS. It will additionally require the setup of a PostgreSQL database.

Setting up Visual Studio Code

We recommend using [Visual Studio Code](#) (VSCode) for its integration with WSL 2, Docker, and the Postgres database. This section assumes you will be using Visual Studio Code for development with Semester.ly.

1. **If you are on Windows OS**, see the following guide on [installing Windows Subsystem for Linux \(WSL\)](#). We recommend choosing Ubuntu 20.04 as your linux distribution. Make sure you take the extra steps to enable WSL 2 as it will be required for Docker.

After WSL 2 is installed, install the [Remote - WSL extension](#) by Microsoft in VSCode. This will allow you to open a VSCode window within your linux subsystem. Press `Ctrl+Shift+P` and select the option `Remote-WSL: New WSL Window`.

2. Install the [Docker extension](#) by Microsoft, the [remote containers extension](#) by Microsoft and the [Postgres extension](#) by Chris Kolkman.

3. Ensure that you are in a WSL Window in VSCode before continuing to the next step. You can open a terminal by selecting the menu option `Terminal -> New Terminal`.

Fork/Clone The Repository

Forking Semester.ly will create your own version of Semester.ly listed on your GitHub! Cloning your Semester.ly fork will create a directory with all of the code required to run your own local development server. Navigate to the directory you wish to work from, then execute:

1. **Fork** navigate to our [GitHub repository](#) then, in the top-right corner of the page, click Fork.
2. **Clone** by executing this line on the command line:

Note: ATTENTION: Be sure to replace [YOUR-USERNAME] with your own git username

```
git clone https://github.com/[YOUR-USERNAME]/semesterly
```

3. Enter the directory:

```
cd semesterly
```

4. Set up the upstream remote to `jhuopensource/semesterly`:

```
git remote add upstream https://github.com/jhuopensource/semesterly
```

Setting up Docker

Steps are below on getting your local development environment running:

1. **Download and install docker for your environment (Windows/Mac/Linux are supported)** <https://www.docker.com/get-started>
2. Create `semesterly/local_settings.py` as follows:

```
DEBUG = True
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
```

(continues on next page)

(continued from previous page)

```

        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': '',
        'HOST': 'db',
        'PORT': '5432',
    }
}

```

Note: ATTENTION: When you clone the repo, you get a folder called semesterly and inside there is another folder called semesterly. Put this in the second semesterly folder.

3. Edit `semesterly/dev_credentials.py` and add a value for `JHU_API_KEY` in single quotes like below.

You can request this API KEY from <http://sis.jhu.edu/api>.

```
'JHU_API_KEY': 'xxxxxxxx',
```

Note: ATTENTION: This is also in the second semesterly directory.

Now run this command in your terminal to make sure that this file isn't tracked by Git and your API key stays local to you.

```
git update-index --skip-worktree semesterly/dev_credentials.py
```

Alternatively, you may create `semesterly/sensitive.py` as follows:

```

SECRETS = {
    'JHU_API_KEY': 'xxxxxxxx',
    # Other sensitive information goes here
}

```

This file will automatically be ignored by git. Be sure to replace 'xxxxxxxx' with your own API key.

4. **Append** this entry to your hosts file as follows (This file is in `C:\Windows\System32\drivers\etc\hosts` or `/etc/hosts`)

```
127.0.0.1      sem.ly jhu.sem.ly
```

Note: ATTENTION: If you're working on other schools, add their URLs here as well (i.e. `uoft.sem.ly` for University of Toronto).

5. Launch terminal or a command window and run:

```
docker-compose build && docker-compose up
```

The **build** command creates a local database and build of your source code. The **up** command runs everything. Be careful not to build when you don't need to as this will destroy your entire database and you'll need to ingest/digest again to get your course data (which takes about 30 minutes).

Note: If you run into additional errors, try the following:

1. Change “buildkit” from true to false in Settings -> Docker Engine.
 2. Refer to the [Docker troubleshooting document](#)
-

Open a browser and visit <http://jhu.sem.ly:8000> to verify you have Semester.ly running.

Note: In order to log in on your local running version of Semester.ly, you will need access to auth keys. Please ask one of the current developers for access to these keys if you require use of login authentication for development. Furthermore, some logins require use of https, so ensure that you are on <https://jhu.sem.ly> instead of <http://jhu.sem.ly:8000> in these cases.

Tip: If you ever need to hard reset Docker, use the command `docker system prune -a`. You can then follow up with `docker-compose build && docker-compose up`.

Setting up Postgres

You can easily access the Postgres database within VSCode by following the next steps. You should have the [Postgres extension](#) by [Chris Kolkman](#) installed.

1. Open the Postgres explorer on the left pane and click the plus button in the top right of the explorer to add a new database connection.
2. Enter `127.0.0.1` as the database connection.
3. Enter `postgres` as the user to authenticate as.
4. Enter nothing as the password of the PostgreSQL user.
5. Enter `5432` as the port number to connect to.
6. Select `Standard Connection`.
7. Select `postgres`.
8. Enter a display name for the database connection, such as `semesterly`.

Upon expanding a few tabs under the new `semesterly` database, you should see several tables. Right clicking any of these tables gives you options to select (view) the items in the table or run a query.

If this is your first time running Semester.ly, you will want to populate your database with courses. Before you continue to [Loading the Database](#), please read the following additional tips for working with Docker and Postgres.

1.3.2 Loading the Database

To load the database you must ingest (create the course JSON), validate (make sure the data makes sense), and digest (load the JSON into the database). You can do so using the following commands:

Tip: You will often have to run commands within the Docker containers. To access containers, open the Docker explorer on the left pane. There should be three containers named `jhuopensource/semesterly`, `semesterly`, and `postgres:12.1`. Right clicking any of these should give you the option `Attach Shell`, which will open a terminal into the corresponding container. For this section, attach the shell to the `semesterly` container.

Ingest

Note: To parse JHU data, you will need to acquire an API access key from [SIS](#). Add the key to `dev_credentials.py` in the `semesterly/` directory. Also, note that the `[SCHOOLCODE]` is `jhu`.

```
python manage.py ingest [SCHOOLCODE] --years [YEARS] --terms [TERMS]
```

For example, use `python manage.py ingest jhu --years 2022 --terms Spring` to parse Spring 2022 courses. You may also leave out the school code to parse all schools. This will run for a substantial amount of time and is not recommended.

Note: If you have ingested before and still have the JSON file on your device, you may skip ingesting and simply digest the old data. This is useful if you are resetting your database during development and wish to quickly reload course data.

Digest

```
python manage.py digest [SCHOOLCODE]
```

You may leave out the school code to digest all schools.

Learn More & Advanced Usage

There are advanced methods for using these tools. Detailed options can be viewed by running

```
python manage.py [command] --help
```

If you are developing a parser or contributing to the pipeline design, you will more than likely need to learn more. Checkout [Data Pipeline Documentation](#) or [Add a School](#)

Tip: You may need to run Postgres commands beyond what running queries through the Postgres extension is capable of. In this case, attach a shell to the postgres container and run `psql -U postgres`. You should now be in the postgres shell. You can use `\q` to leave it.

1.3.3 Advanced Configuration

VSCoDe Extensions

Tip: Previously in [Installation](#), we told you to install the [remote containers extension](#) by Microsoft. When you `docker-compose up`, in the Docker tab when right-clicking a container, you should see [Attach Visual Studio Code](#) in addition to [Attach Shell](#). It is recommended that you develop (and install these extensions) while using VSCode attached to the container in order to match the build environment.

Extensions can help you be more productive when working on Semester.ly code. Feel free to ask current developers what extensions they use. Here are a few we suggest:

1. **Python + PyLance.** With this extension, you can set your default formatter (black) and default linter (pycodestyle). If you choose to set pycodestyle as your linter, be sure to change max-line-length to 88.
2. **JavaScript + TypeScript.** TypeScript support for development.
3. **ESLint.** As one of our checks requires ESLint to be satisfied, this will save you some time.
4. **Prettier.** Formats JS/TS for you. You will want to set Prettier as your default formatter, and we suggest you set Format Document On Save to be on in your VSCode preferences.
5. **IntelliCode.** Provides useful suggestions.
6. **GitHub Copilot.** Can often write your code for you, but be sure to double check it.
7. **Bookmarks.** Lets you mark places in code that you want to revisit.
8. **Rewrap.** It helps with wrapping text for you when editing documentation or code comments.
9. **Sourcery.** Sometimes will help you write cleaner Python code.
10. **SpellChecker.** Helps you find typos in documentation.

Overriding/Setting Secrets

Note: This step is not necessary for most developers. Only continue reading this section if you need to override the test secrets (API keys/credentials) provided by Semester.ly (which are for testing only).

Semester.ly makes use of several secrets which allow it to interact securely with third party software providers. These providers include Facebook (for oauth and social graph), Google (oauth), and university APIs.

In order for Semester.ly to run out of the box, we have included credentials to test Google and Facebook applications for development purposes. We override these keys for production use thereby keeping our client secrets... well, secrets! These provided credentials can be found in `semesterly/dev_credentials.py`:

```
SECRETS = {
    #Credentials for a test application for Semester.ly (+ Google/Facebook)
    'SECRET_KEY': ...,
    'HASHING_SALT': ...,
    'GOOGLE_API_KEY': ...,
    'SOCIAL_AUTH_GOOGLE_OAUTH2_KEY': ...,
    'SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET': ...,
    'SOCIAL_AUTH_FACEBOOK_KEY': ...,
    'SOCIAL_AUTH_FACEBOOK_SECRET': ...,
    'FB_TEST_EMAIL': ...,
    'FB_TEST_PASS': ...,
    'SOCIAL_AUTH_AZURE_TENANT_KEY': ...,
    'SOCIAL_AUTH_AZURE_TENANT_SECRET': ...,
    'SOCIAL_AUTH_AZURE_TENANT_ID': ...,
    'STUDENT_SIS_AUTH_SECRET': ...,

    #Not essential for testing, but can be filled in for advanced usage
    ...
}
```

However, if you wish to override these credentials or add login credentials for a school which requires a client secret, you may add your key/value pair to `semesterly/sensitive.py`. This file is gitignored and will be kept private so

you can safely store the private information you wish within this file. It should have a format identical to SECRETS above and in `semesterly/dev_credentials.py`.

Using Secrets

In order to properly access a secret from anywhere within the code, simply import the `get_secret` function and use it to access the secret by key:

```
from semesterly.settings import get_secret
hashids = Hashids(salt=get_secret('HASHING_SALT'))
```

This will check the following locations for the secret (in order, using the first value it finds), throwing an error if it does not find the key at all:

1. Check OS environment variables
2. Check `semesterly/sensitive.py`
3. Default to `semesterly/dev_credentials.py`
4. Error

1.3.4 High Level Design

A high level description of what Semester.ly is, how it works, and which parts do what

Problem

Course registration is a complicated process involving several factors that influence which courses a student decides to take, such as degree requirements, professor and course ratings, friends, and personal interests. Trying to keep track of potential schedules a student can take can easily become overwhelming.

Solution

Semester.ly aims to make course registration easy and collaborative through a user interface that focuses on student needs, providing quick access to necessary tools a student may need in order to organize and decide on what courses they want to take.

Current Features

Search

The screenshot displays the Semester.ly search interface. At the top left is the Semester.ly logo. A search bar contains the text 'Fall 2022 | fiction'. Below the search bar, three search results are listed:

- Introduction to Fiction & Poetry I (AS.220.105) with a red letter 'A' next to it.
- Introduction to Fiction & Poetry II (AS.220.106)
- The Craft of Fiction: Narrative Perch (AS.220.200)

Each result has a bookmark icon and a plus sign icon. A tooltip for the first result is open, showing a grid of 26 lecture sections. Section (26) is highlighted with a red 'D'. The background shows a timetable grid for Monday (Mon) with time slots from 8:00 to 12:00.

By typing in a query in the search field at the top of the site, students can quickly search for courses that they are looking for.

- A. Clicking on the course search result will reveal more information about the course (see below).
- B. [Deprecated] This button will add the course to your optional courses, meaning Semester.ly will try to fit the course into your schedule if there's space for it.
- C. Add course to schedule.
- D. Hovering over these course sections will preview what your schedule looks if you were to add the course.

Introduction to Fiction & Poetry I

AS.220.105, Writing Seminars H Writing Intensive

3

credits

Average Course Rating

★★★★★

50% of Seats Added on Semesterly

Prerequisites

None

Program of Study Tags

None

Friends in This Course

No Classmates Found

Friends Who Have Taken This Course

No Classmates Found

Reactions

Check out your classmate's reactions – click an emoji to add your own opinion!

🔥 12
❤️ 12
🍕 7
😬 8
😬 5
😬 5
100 12
🤖 6

Course Description

An introduction to basic strategies in the writing of poetry and fiction, with readings by Joyce, Woolf, Baldwin, Munro, Garcia Marquez, Donne, Bishop, Yeats, Komunyakaa, Tretheway, and others. Students will learn the elements of the short story and try their hand at a variety of forms: realist, fantastical, experimental. They'll also study the basic poetic forms and meters, from the ballad to the sonnet, iambic pentameter to free verse. Students will compose short stories and poems and workshop them in class. This course is a prerequisite for most upper level courses. This course is part one of the year-long Introduction to Fiction and Poetry, and must be taken before [AS.220.106](#).

Course Evaluations

Spring 2015 (Emet) ★★★★★ (4.71)	Spring 2015 (Poetry) ★★★★★ (3.93)	Spring 2015 (Landry) ★★★★★ (4.5)
Spring 2015 (Eisman) ★★★★★ (3.86)	Spring 2015 (Xie) ★★★★★ (4.6)	Spring 2015 (Frantz) ★★★★★ (4.79)
Spring 2015 (y) ★★★★★ (4.33)	Spring 2015 (Booe) ★★★★★ (4.25)	Spring 2015 (Goldberg) ★★★★★ (4.5)
Spring 2015 (DoI) ★★★★★ (4.33)	Spring 2015 (Siskel) ★★★★★ (4.4)	Spring 2015 (Childers) ★★★★★ (3.69)
Spring 2015 (Koeckoeck) ★★★★★ (4.86)	Spring 2015 (Hudgins) ★★★★★ (3.67)	Spring 2015 (Raskulinecz) ★★★★★ (3.93)
Spring 2015 (Winchester) ★★★★★ (4.86)	Spring 2015 (Daynes) ★★★★★ (4.86)	Fall 2014 (Stintzi) ★★★★★ (4.86)

Lecture Sections (Hover to see the section on your timetable)

(01) R. Hubbell 2 waitlist / 15 seats	(02) J. Cox 0 open / 15 seats
(03) R. Hubbell 1 waitlist / 15 seats	(04) E. Emmons 0 open / 15 seats
(05) C. Wray 0 open / 15 seats	(06) C. Atherton 1 waitlist / 15 seats
(07) H. Choi 1 waitlist / 15 seats	(08) E. Emmons 0 open / 15 seats
(09) D. Guida 1 waitlist / 15 seats	(10) D. Carpenter 2 waitlist / 15 seats
(11) M. Cook 0 open / 15 seats	(12) B. Kessler 0 open / 15 seats
(13) B. Kessler 6 waitlist / 15 seats	(14) S. Neugebauer 3 waitlist / 15 seats
(15) K. Ugwueze 3 waitlist / 15 seats	(16) D. Carpenter 2 waitlist / 15 seats
(17) L. Raszick 0 open / 15 seats	(18) S. Neugebauer 0 open / 15 seats
(19) K. Ugwueze 1 waitlist / 15 seats	(20) B. Steidle 2 waitlist / 15 seats
(21) L. Raszick 4 waitlist / 15 seats	(22) B. Basham 1 open / 15 seats

Example of course information, which includes how many credits the course is, a brief description, [deprecated] course evaluations, sections, and students' reactions

There is also an option for Advanced Search, allowing for filtering of department, area, course level, and day/time.

The screenshot shows the 'Advanced Search' interface. On the left, there are several filter sections: 'Department Filter' with 'Computer Science' selected, 'Area Filter' with 'Q' selected, 'Level Filter' with '200' selected, and 'Day/Time Filter' with 'Mon' and 'Wed' selected. The main content area displays the course details for 'Data Structures EN.601.226'. It includes a share icon, a bookmark icon, and a plus icon. Below these is a large '4 credits' display, followed by an 'Average Course Rating' of 4.5 stars and '41% of Seats Added on Semesterly'. The 'Prerequisites' section lists 'EN.500.132 OR (EN.500.112 or EN.601.220) or AP Computer Science or equivalent'. The 'Program of Study Tags' section lists 'COGS-COMPCG, ROBO-CMCS'. The 'Friends In This Course' section shows 'No Classmates Found'. The 'Reactions' section shows a row of emojis with counts: fire (3), smile with hearts (2), poop (1), neutral (3), sad (1), angry (2), 100 (2), and nerd (6).

The courses you add to your schedule will show up in a color-coded display to allow you to easily distinguish between courses and when they take place.

Scheduling

The screenshot shows the 'Scheduling' interface. At the top, it says 'Semester.ly' and 'Fall 2022 | Searching Fall 2022'. Below this is a weekly calendar grid with columns for Mon, Tue, Wed, Thu, Fri, Sat, and Sun. The grid shows three courses scheduled: 'Introduction to Fiction & Poetry I (01)' on Monday, Wednesday, and Friday. A 'Registered Courses' sidebar on the right shows a list of courses with letters A through P next to them. The courses listed are: 'Ideal', 'Backup', 'AS.220.106 Introduction to Fiction & Poetry II', and 'Optional Courses'.

- Rotate through potential schedules based on differing sections
- Switch the semester from, e.g. Fall 2022 to Spring 2022
- Open the Advanced Search
- Add current courses to SIS cart; requires JHU Login

- E. Add a custom event; this is to add, e.g. an extracurricular to the schedule, or any other activity that is not a course.
- F. Generate a share link to share the schedule with other students
- G. Create a new schedule
- H. Export the calendar to a .ics file
- I. Preferences menu, e.g. toggle on/off weekends
- J. Change schedule name
- K. Select another schedule (of the same semester) to switch to it.
- L. (Behind the dropdown) Find New Friends displays other students who are also taking your classes.
- M. Compares two schedules together, showing same and differing courses
- N. Opens various account settings
- O. Duplicate schedule
- P. Delete schedule

Screenshots

The screenshot displays the Semester.ly interface for comparing two schedules. The main view is a calendar grid with days of the week (Mon-Fri) and times (8:00-9:00). The 'Active' schedule (left) has a total of 17.00 credits, and the 'Compared' schedule (right) has 14.00 credits. The compared schedule is highlighted in green, showing differences in course selection and timing. A list of course details is visible on the right side of the interface.

Day	Time	Active Course	Compared Course
Mon	1:00 - 2:00	First Year Heritage Chinese (02)	First Year Heritage Chinese (02)
Tue	1:00 - 2:00	Fourth Year Korean (01)	Fourth Year Korean (01)
Wed	1:00 - 2:00	First Year Heritage Chinese (02)	First Year Heritage Chinese (02)
Thu	1:00 - 2:00	Fourth Year Korean (01)	Fourth Year Korean (01)
Fri	1:00 - 2:00	First Year Heritage Chinese (02)	First Year Heritage Chinese (02)
Mon	4:30 - 5:45	Impressional Techniques for Communication (01)	Impressional Techniques for Communication (01)
Tue	3:00 - 4:15	Database (01)	Database (01)
Wed	4:30 - 5:45	Impressional Techniques for Communication (01)	Impressional Techniques for Communication (01)
Thu	3:00 - 4:15	Database (01)	Database (01)
Tue	6:15 - 7:30	Financial Accounting (04)	Financial Accounting (04)
Thu	6:15 - 7:30	Financial Accounting (04)	Financial Accounting (04)

An example of what it looks like to compare two schedules.

Features in Development

0. Enhancing Search - display more than 4 results and scroll infinitely when searching for courses regularly
1. Dark Mode - option to toggle between light and dark mode
2. Study Groups - option to message students who are also taking your class to ask if they want to study together or go to class together

Tech Stack

Semester.ly pulls data about courses, ratings, and more from all across the internet. It saves this data into a custom representation within a **Postgres database**. The data is retrieved using a variety of **webscraping, HTML parsing, and information retrieval** techniques which we've built into our own mini-library of utilities. This data is entered into the database via the **Django ORM** (Object-Relational Mapping). The ORM allows us to query the database and create rows using python code as if these rows were objects.

We manipulate and access this same data using Django **views** to respond to any web requests directed to our server. For example, when a user clicks on a course to open the course modal, the browser issues a request asking for the data related to that course. Our Django views respond with a **JSON** representation of the course data for rendering on the UI.

The browser knows when and how to make these requests, as well as how to generate the UI based on the responses using React and Redux. **React and Redux** maintain application state and use **Javascript/Typescript** to render **HTML** based on that state.

Finally, this HTML is styled with **SCSS** for an appealing, cohesively styled user experience!

The Apps that Make Semester.ly

The overall, the Semester.ly application is made up of many smaller *apps* which each handle some collection of logic that makes Semester.ly tick! Each app encapsulates a set of urls which map a request to a view, views which respond to requests with HTML/JSON/etc, models which represent tables in the database, and tests which ensure Functionality behaves as expected.

App Name	Key Models/Functionality	Description
Agreement	Terms of Service and Privacy Policy views	Tracks changes to terms of service and privacy policy.
Analytics	Models: SharedTimetable, Device-Cookie, Feature Views	Tracks analytics on the usage of features as objects in the database. Renders a dashboard at <code>/analytics</code> .
Authpipe	Authentication, login, signup	Authentication pipeline functions for the authentication of users, creation of students, and loading of social data.
Courses	Course Serializer, Views for returning course info	Functionality for accessing course data, the course modal, course pages
Integrations	Integration views	Functionality for integrating school specific code to appear in search or in the course modal
Parsing	Scrapers, parsers, parsing utilities	Home of the data pipeline that fills our database
Searches	Advanced search, basic search	Views for parsing queries and returning course data
Semesterly	No core models, views, or functionality; contains Django settings.	Delegates urls to sub-apps, contains end-to-end tests, other configuration.
Students	Models: Student, Personal Timetables, Reactions, Personal Event	All logic for logged-in specific users. Creating and saving a personal timetable, reacting to courses, saving custom events.
Timetable	Models: Course, Section, Offering, Timetable, Semester, Evaluations	Timetable generation and all models required for timetable representation.

1.3.5 Learning The Stack

Note: Learning a new thing can be scary, especially when all you have are some docs and a massive code base to learn from. That's why we are here to help you learn, build, and contribute. Ask us questions at our [Discord!](#)

Our Stack

Component	Technology	Style/Methodology
Database	PostgreSQL	Django ORM
Backend Framework	Django	pycodestyle/Black
Frontend Framework	React/Redux	ESLint/Prettier
CSS Framework	SCSS	BEM/Airbnb

Tutorials and Resources

Learning the Backend

Django is a Python Web framework that provides a huge number of tools for web developers to quickly write scalable code with minimal configuration. It is used all over the tech industry by companies like Spotify, Instagram, YouTube, and DropBox!

[Writing your first Django app](#) is the official Django tutorial. It is top notch! The official documentation can be found at the same url and provides high quality information about how to build with this modern web framework. For example, here's the documentation on [making queries with Django](#).

Learning React/Redux

React is a Javascript library created by Facebook for building user interfaces. It allows developers to make encapsulated components that can be written once and used anywhere.

Redux is state container that makes React development easier to manage long term!

The official docs are the go-to: [React Basics](#), [React Hooks](#), [Redux & Redux Toolkit](#), and [TypeScript](#). We suggest going through the step-by-step React tutorial rather than the practical tutorial because the practical tutorial is done with class-based components, but we prefer functional components.

If you're looking for something more structured, one suggestion is this [Udemy course](#), which covers functional React, Redux concepts, TypeScript, and more.

Ultimately, the best practice will be to create a small project using `npx create-react-app my-app --template redux-typescript` and going from there. You will become much more familiar with all of the concepts when you try to work through it yourself.

Learning CSS/SCSS

The most important step is to [learn the CSS basics](#).

With that, you can [dive into SCSS](#), a css preprocessor.

For development, we use the BEM methodology ([learn about BEM here!](#)) and the [Airbnb style guide](#).

Learning Scraping/Parsing

Coming soon!

1.3.6 How to Contribute

Contributing to Semester.ly follows the following simple workflow:

1. *Create a Branch*
2. *Make Changes*
3. *Clean Up Changes*

Create a Branch

Make sure you have followed all of the instructions in [Installation](#) to set up your local repository and upstream remote.

We follow the [Gitflow workflow](#); our main branch is `prod`, and our develop branch is `develop`. The general gist is that for anything new, you want to branch off of `develop` and name your branch `feature/your-branch-name`. Two other conventions we have is for bug fixes we use `fix/your-branch-name`, and for refactoring we use `refactor/your-branch-name`. In the case you need to fix something that was just released, and it needs to go straight to production, then branch off of `prod` and name your branch `hotfix/your-branch-name`.

To stay up to date with upstream/develop, you'll want to `git pull` whenever you're starting a new branch. You may need to `git fetch upstream` first.

```
git checkout develop
git pull upstream develop
```

Then, you'll want to create a new branch.

```
git checkout -b <your-branch-name>
```

Make Changes

After you've made edits, `git add` your files, then commit. One way to do this:

```
git add <path_to_file>
git commit -m "Topic: Message"
git push --set-upstream origin your-branch-name
```

Note: It is preferred that you follow the commit message convention of "Topic: Message". This helps when we are browsing through commits so we can quickly identify what each commit was about. **Messages should be in the imperative mood**, as if you're telling someone what to do. If it helps, you are encouraged to include the how/why -

“*Evaluation list: Duplicate state to avoid modifying redux state*”. Furthermore, try to keep commits to “one” change at a time and commit often.

From here, you should be prompted to create a new pull request (PR). Ctrl + Left Click to open the link. From there, add a short description on what your PR does and how/why you did it, and then create the PR. If your PR is ready for review, add a reviewer as well.

Note: What If Upstream Has Changed? If merging upstream into your branch does not cause any conflicts, using rebase is a good option.

```
git pull --rebase upstream develop
git push origin your-branch-name
```

However, if there are merge conflicts, I suggest creating an alternate branch off of your branch and then merging upstream, fixing any conflicts, and then merging back into your branch. Although more complicated, this saves you from messing up the work on your branch if the merge conflicts aren’t easily resolved, or you make a mistake while resolving the conflicts.

```
git checkout develop
git pull upstream
git checkout your-branch-name
git checkout -b merge-develop
git merge develop
(Fix merge conflicts, git add + git commit)
git checkout your-branch-name
git merge merge-develop
git push
```

Clean Up Changes

We have GitHub workflows that check your changes and run them against our automated tests. While the workflow is building, we have a few other workflows that check the style and formatting of your code, and they will run more quickly than the build flows. Take this time to fix any formatting or linting issues should these tests fail. Refer to the [Style Guide](#) to learn more about our code guidelines.

Note: A PR must pass a few checks before it can be merged.

LGTM: Before your PR is merged, you’ll need to pass a peer review to ensure that all the changes are clean and high quality. Usually, you’ll get an “LGTM” or a few minor edits will be requested. This helps us maintain a quality code base and helps contributors learn and grow as engineers!

PR Body: Your pull request should reference a git issue if a related issue has been created. Additionally, it must provide an in depth description of why the changes were made, what they do, and how they do it.

Tests & Builds Pass: All tests and builds, as run by Github Actions, must pass.

Linting Satisfied: All files must successfully pass our code style checks.

```
npx prettier "**/*.{js,jsx,ts,tsx}" --write
eslint . --ext .js,.jsx,.ts,.tsx --fix
black .
```

1.3.7 Style Guide

Javascript/Typescript Style Guide

1. We follow the ESLint style guideline as configured in our `.eslintrc.js` file.
2. We format our frontend code using Prettier. Line length is configured to 88, like the backend.
3. Use PascalCase for React components, camelCase for everything else.
4. When creating new components, make them **functional** components.
5. When adding Redux state, make them **slices**.

As you can see, we are shifting towards using functional React components and Redux Toolkit with TypeScript. As such, all new features are expected to use these technologies. If you find that you can achieve what you are trying to do more easily by refactoring a class-based component to a functional component, or a reducer to a slice, you are encouraged to do so.

Python Style Guide

1. We follow the pycodestyle style guide for Python, with the exception that the max-line-length is 88 instead of 79. This is to comply with the default settings of the autoformatter black.
2. Use snake_case for variable names and functions. Use PascalCase for classes. Use f-strings over %s strings or `.format()`.
3. Use type annotations when the type of a variable is ambiguous.
4. If possible, helper functions go *after* the function they appear in. Do not put them before the method as is commonly done in languages like C.

1.3.8 Add a School

Adding a new school is easy and can be done in a few simple steps:

1. *Run the Scaffolder*
2. *Develop the Parser*
3. *Parse and Test*

Run the Scaffolder

Running the `makeschool` command will create a directory for your school, creating a configuration file, a stub for the parser, etc. Run the following for your school:

```
python manage.py makeschool --name "University of Toronto" --code "uoft" --regex "([A-Z]
↪{2,8}\\s\\d{3})"
```

Don't forget to add this new school to your `/etc/hosts`! (Check here for a reminder on how: [Installation](#))

Develop the Parser

Note: Notify us if you intend to add a school! Create a GitHub issue with the tag `new_school`. We can help you out and lend a hand while also keeping track of who's working on what!

The scaffolder created the stub of your parser. It provides the start function and two outer loops that iterate over each provided term and year. **Your goal is to fill the inside of this so that for each year and term, you collect the course data for that term/year.**

What this boils down to is the following template:

```
for year in years:
    for term in terms:

        departments = get_departments(term, year)

        for department in departments:

            courses = get_courses(department)

            for course in courses:
                self.ingestor['course_code'] = ...
                self.ingestor['department'] = ...
                self.ingestor['description'] = ...
                ...
                self.ingestor.ingest_course()

            for section in sections:
                self.ingestor['section_code'] = ...
                self.ingestor['section_type'] = ...
                self.ingestor['year'] = ...
                self.ingestor['term'] = ...
                ...
                self.ingestor.ingest_section()

            for meeting in meetings:
                ...
                self.ingestor.ingest_meeting()
```

Breaking it down

The code starts out by getting the departments. It doesn't have to, but often it is easiest to go department by department. The parser then collects the courses for that department. We will talk about how it does this in *How To Fill The Ingestor*.

For each course, the parser fills the ingestor with the fields related to the course (e.g. description, the course code). Once complete, it calls `ingest_course` to execute the creation of the course.

It then repeats this process for the sections belonging to that course, and for each section, the meetings (individual meeting times) belonging to the section.

Everything else is handled by the BaseParser and the ingestor for you.

How To Fill The Ingestor

As shown by the code sample above, filling the ingestor is as easy as filling a python dictionary. The only question that remains is how to collect the data to fill it with.

The answer is by pulling it from the internet of course! Luckily we have a tool called the **Requester** which helps developers like you to *request* information from a web course catalogue or API.

Using the Requester

By inheriting from the BaseParser, your parser comes with its own requester that can be used like this:

```
markup = self.requester.get('www.siteorapi.com')
```

or:

```
markup = self.requester.post('www.siteorapi.com', data=form)
```

It will automatically return a marked-up version of the data returned by the request (automatically detecting JSON/XML/HTML).

Note: The requester will maintain a [session](#) for you, making sure the proper cookies are stored and sent with all future requests. It also [randomizes the user agent](#). Future updates will automatically parallelize and throttle requests (*a great project to contribute to the data pipeline*).

Parsing JSON

In the event that your source of course data returns JSON, life is easy. You can find the fields and pull them out by simply treating the JSON as a python dictionary when the requester returns it.

Parsing HTML (or XML)

If, instead, your site is marked up with HTML, we use [BeautifulSoup4 \(BS4\)](#) to find certain divs and map the data inside of those divs to the fields of the ingestor.

Let's say the HTML looks like this:

```
<body>
  <div class="course-wrapper">
    <h1>EN.600.123</h1>
    <h4>Some Course Name</h4>
    <a href="urltosectiondata">More Info</a>
    ....
  </div>
  <div class="course-wrapper">
    ...
  </div>
  ...
</body>
```

We can then write the get courses function as follows:

```
def get_courses(self, department):
    soup = self.requester.get('urltothisdepartment.com')
    return soup.find_all(class_='course-wrapper')
```

And we can fill the ingestor based on these courses by:

```
courses = self.get_courses(department)
for course in courses:
    self.ingestor['course_code'] = course.find('h4').get_text()
    ...
```

To get section data, we can follow the “More Info” link and parse the resulting HTML in the same way:

```
section_html = self.requester.get(course.find('a')['href'])
```

Note: You can learn more about BS4 by [reading their documentation](#) . It is an extensive library that provides many excellent utilities for parsing HTML/XML.

Parse and Test

When you’re ready you can go ahead and run your parser. You can do this by:

```
python manage.py ingest [SCHOOL_CODE]
```

Replacing SCHOOL_CODE with whatever your school’s code (e.g. `jhu`) is. This will start the ingestion process, creating a file `data/courses.json` in your school’s directory.

If, along the way, your ingestion fails to validate, the ingestor will throw useful errors to let you know how or why!

Once it runs to completion, you can *digest* the JSON, entering it into the database by running:

```
python manage.py digest [SCHOOL_CODE]
```

Note: To learn more, checkout the [Data Pipeline Documentation](#)

1.3.9 How to Run & Write Tests

Running Tests

Frontend

Run all tests:

```
npm test
```

Run single test:

```
npm test -- static/js/redux/__tests__/schema.test.js
```

Backend

Run all tests:

```
python manage.py test
```

Run all tests for a single app:

```
python manage.py test timetable
```

Run single test suite:

```
python manage.py test timetable.tests.UrlTest
```

Run single test case:

```
python manage.py test timetable.tests.UrlTest.test_urls_call_correct_views
```

Run tests without resetting db:

```
python manage.py test -k
```

Our current test runner will only run db setup if the tests you're running touch the db.

Writing Tests

Unit Tests

Contributors are encouraged to write unit tests for changed and new code. By separating out logic into simple pure functions, you can isolate the behavior you care about in your unit tests and not worry about testing for side effects. Following the design principles outlined in the resources from the *Learning The Stack* section helps with this. For example, extracting all code that extract information from the state into selectors, which are pure functions that take the state (or some part of it) as input and output some data, will make it easy to test and change state-related behavior. Sometimes you may want to test behavior that can't be extracted into a pure function or that touches external interfaces. There are a number of strategies you can use in these cases.

Integration Tests

In the frontend, for testing the logic for rendering a component, look into snapshot tests. For testing async (thunk) action creators, our current tests create a store with desired initial state, dispatch the action, and then check that the action had the desired effect on the state. Backend requests are mocked using the *nock* library.

For testing views, we use django's built-in client to send requests to the backend. It's also possible to use django's request factory to create requests to provide directly as input to your views.

End to End Tests

As the name implies, end to end tests test the entire app at once by simulating a semesterly user. When writing or changing end to end tests, it is recommended to familiarize yourself with the methods provided in `SeleniumTestCase`, which make it easy to perform certain actions on the app.

1.3.10 Backend Documentation

Timetable App

The timetable app is the core application that has been a part of Semester.ly since our very first release. The timetable app does the heavy lifting for timetable generation, sharing, and viewing.

Models

class `timetable.models.Course(*args, **kwargs)`

Represents a course at a school, made unique by its course code. Courses persist across semesters and years. Their presence in a semester or year is indicated by the existence of sections assigned to that course for that semester or year. This is why a course does not have fields like professor, those varies.

The course model maintains only attributes which tend not to vary across semesters or years.

A course has many `Section` which a student can enroll in.

school

this course's school's code

Type CharField

code

the course code without indication of section (e.g. EN.600.100)

Type CharField

name

the general name of the course (E.g. Calculus I)

Type CharField

description

the explanation of the content of the course

Type TextField

notes

usually notes pertaining to registration (e.g. Lab Fees)

Type TextField, optional

info

similar to notes

Type TextField, optional

unstopped_description

automatically generated description without stopwords

Type TextField

campus

an indicator for which campus the course is taught on

Type CharField, optional

prerequisites

courses required before taking this course

Type TextField, optional

corequisites

courses required concurrently with this course

Type TextField, optional

exclusions

reasons why a student would not be able to take this

Type TextField, optional

num_credits

the number of credit hours this course is worth

Type FloatField

areas

list of all degree areas this course satisfies.

Type Arrayfield

department

department offering course (e.g. Computer Science)

Type CharField

level

indicator of level of course (e.g. 100, 200, Upper, Lower, Grad)

Type CharField

cores

core areas satisfied by this course

Type CharField

geneds

geneds satisfied by this course

Type CharField

related_courses

courses computed similar to this course

Type ManyToManyField of *Course*, optional

same_as

If this course is the same as another course, provide Foreign key

Type ForeignKey

exception DoesNotExist

exception MultipleObjectsReturned

get_avg_rating()

Calculates the avg rating for a course, -1 if no ratings. Includes all courses that are marked as the same by the self.same_as field on the model instance.

Returns the average course rating

Return type (`float`)

get_reactions(*student=None*)

Return a list of dicts for each type of reaction (by title) for this course. Each dict has:

title: the title of the reaction

count: number of reactions with this title that this course has received

reacted: True if the student provided has given a reaction with this title

class `timetable.models.CourseIntegration`(*id, course, integration, json*)

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `timetable.models.Evaluation`(**args*, ***kwargs*)

A review of a course represented as a score out of 5, a summary/comment, along with the professor and year the review is in subject of.

course

the course this evaluation belongs to

Type `ForeignKey` to `Course`

score

score out of 5.0

Type `FloatField`

summary

text with information about why the rating was given

Type `TextField`

professor

the professor(s) this review pertains to

Type `CharField`

year

the year of the review

Type `CharField`

course_code

a string of the course code, along with section indicator

Type `Charfield`

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `timetable.models.Integration`(*id, name*)

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class timetable.models.**Offering**(*args, **kwargs)

An Offering is the most granular part of the Course heirarchy. An offering may be looked at as the backend equivalent of a single slot on a timetable. For each day/time which a section meets, an offering is created.

section

the section which is the parent of this offering

Type ForeignKey to [Section](#)

day

the day the course is offered (single character M,T,W,R,F,S,U)

Type CharField

time_start

the time the slot starts in 24hrs time in the format (HH:MM) or (H:MM)

Type CharField

time_end

the time it ends in 24hrs time in the format (HH:MM) or (H:MM)

Type CharField

location

the location the course takes place, defaulting to TBA if not provided

Type CharField, optional

exception DoesNotExist

exception MultipleObjectsReturned

class timetable.models.**Section**(*args, **kwargs)

Represents one (of possibly many) choice(s) for a student to enroll in a [Course](#) for a specific semester. Since this model is specific to a semester, it contains enrollment data, instructor information, etc.

A section can come in different forms. For example, a lecture which is required for every student. However, it can also be a tutorial or practical. During timetable generation we allow a user to select one of each, and we can automatically choose the best combination for a user as well.

A section has many offerings related to it. For example, section 1 of a [Course](#) could have 3 offerings (one that meets each day: Monday, Wednesday, Friday). Section 2 of a [Course](#) could have 3 other offerings (one that meets each: Tuesday, Thursday).

course

The course this section belongs to

Type [Course](#)

meeting_section

the name of the section (e.g. 001, L01, LAB2)

Type CharField

size

the capacity of the course (the enrollment cap)

Type IntegerField

enrolment

the number of students registered so far

Type IntegerField

waitlist
the number of students waitlisted so far
Type IntegerField

waitlist_size
the max size of the waitlist
Type IntegerField

section_type
the section type, example 'L' is lecture, 'T' is tutorial, *P* is practical
Type CharField

instructors
comma separated list of instructors
Type CharField

semester
the semester for the section
Type ForeignKey to *Semester*

was_full
whether the course was full during the last parse
Type BooleanField

course_section_id
the id of the section when sending data to SIS
Type IntegerField

exception DoesNotExist

exception MultipleObjectsReturned

class timetable.models.**Semester**(*args, **kwargs)

Represents a semester which is composed of a name (e.g. Spring, Fall) and a year (e.g. 2017).

name
the name (e.g. Spring, Fall)

Type CharField

year
the year (e.g. 2017, 2018)

Type CharField

exception DoesNotExist

exception MultipleObjectsReturned

Views

class `timetable.views.TimetableLinkView(**kwargs)`

A subclass of `FeatureFlowView` (see [Flows Documentation](#)) for the viewing of shared timetable links. Provides the logic for preloading the shared timetable into `initData` when a user hits the corresponding url. The frontend can then act on this data to load the shared timetable for viewing.

Additionally, on POST provides the functionality for the creation of shared timetables.

get_feature_flow(*request, slug*)

Overrides `FeatureFlowView` `get_feature_flow` method. Takes the slug, decrypts the hashed database id, and either retrieves the corresponding timetable or hits a 404.

post(*request*)

Creates a `SharedTimetable` and returns the hashed database id as the slug for the url which students then share and access.

class `timetable.views.TimetableView(**kwargs)`

This view is responsible for responding to any requests dealing with the generation of timetables and the satisfaction of constraints provided by the frontend/user.

post(*request*)

Generate best timetables given the user's selected courses

Serializers

class `timetable.serializers.DisplayTimetableSerializer(*args, **kwargs)`

class `timetable.serializers.EventSerializer(*args, **kwargs)`

class `timetable.serializers.PersonalTimeTablePreferencesSerializer(*args, **kwargs)`

class `timetable.serializers.SlotSerializer(*args, **kwargs)`

Utils

class `timetable.utils.DisplayTimetable(slots, has_conflict, show_weekend, name="", events=None, id=None)`

Object that represents the frontend's interpretation of a timetable.

classmethod `from_model(timetable)`

Create `DisplayTimetable` from `Timetable` instance.

class `timetable.utils.Slot(course, section, offerings, is_optional, is_locked)`

course

Alias for field number 0

is_locked

Alias for field number 4

is_optional

Alias for field number 3

offerings

Alias for field number 2

section

Alias for field number 1

class `timetable.utils.Timetable(courses, sections, has_conflict)`

courses

Alias for field number 0

has_conflict

Alias for field number 2

sections

Alias for field number 1

`timetable.utils.add_meeting_and_check_conflict(day_to_usage, new_meeting, school)`

Takes a `@day_to_usage` dictionary and a `@new_meeting` section and returns a tuple of the updated `day_to_usage` dict and a boolean which is True if conflict, False otherwise.

`timetable.utils.can_potentially_conflict(course_1_date_start, course_1_date_end, course_2_date_start, course_2_date_end)`

Checks two courses start & end dates to see whether they can overlap and hence potentially conflict. If any of the values are passed as None it will automatically consider that they can potentially conflict. Input type is string but has to be in a reasonable date format.

Parameters

- `{[string]}` -- [course 1 start date in a reasonable date format] (*course_1_date_start*) –
- `{[string]}` -- [course 1 end date in a reasonable date format] (*course_1_date_end*) –
- `{[string]}` -- [course 2 start date in a reasonable date format] (*course_2_date_start*) –
- `{[string]}` -- [course 2 end date in a reasonable date format] (*course_2_date_end*) –

Returns [bool] – [True if if dates ranges of course 1 and 2 overlap, otherwise False]

`timetable.utils.courses_to_slots(courses, locked_sections, semester, optional_course_ids)`

Return a list of lists of Slots. Each Slot sublist represents the list of possibilities for a given course and section type, i.e. a valid timetable consists of any one slot from each sublist.

`timetable.utils.find_slots_to_fill(start, end, school)`

Take a `@start` and `@end` time in the format found in the coursefinder (e.g. 9:00, 16:30), and return the indices of the slots in that array which represents times from 8:00am to 10pm that would be filled by the given `@start` and `@end`. For example, for uoft input: '10:30', '13:00' output: [5, 6, 7, 8, 9]

`timetable.utils.get_current_semesters(school)`

List of semesters ordered by academic temporality.

For a given school, get the possible semesters ordered by the most recent year for each semester that has course data, and return a list of (semester name, year) pairs.

`timetable.utils.get_day_to_usage(custom_events, school)`

Initialize `day_to_usage` dictionary, which has custom events blocked out.

`timetable.utils.get_hour_from_string_time(time_string)`

Get hour as an int from time as a string.

`timetable.utils.get_hours_minutes(time_string)`
Return tuple of two integers representing the hour and the time given a string representation of time. e.g. '14:20'
-> (14, 20)

`timetable.utils.get_minute_from_string_time(time_string)`
Get minute as an int from time as a string.

`timetable.utils.get_time_index(hours, minutes, school)`
Take number of hours and minutes, and return the corresponding time slot index

`timetable.utils.get_xproduct_indicies(lists)`
Takes a list of lists and returns two lists of indicies needed to iterate through the cross product of the input.

`timetable.utils.slots_to_timetables(slots, school, custom_events, with_conflicts, show_weekend)`
Generate timetables in a depth-first manner based on a list of slots.

`timetable.utils.update_locked_sections(locked_sections, cid, locked_section, semester)`
Take cid of new course, and locked section for that course and toggle its locked status (ie if was locked, unlock and vice versa).

Courses App

The courses app deals with the accessing course information, the sharing of courses, and the rendering of the course/all course pages.

Views

class `courses.views.CourseDetail(**kwargs)`
View that handles individual course entities.

get(`request, sem_name, year, course_id`)
Return detailed data about a single course. Currently used for course modals.

class `courses.views.CourseModal(**kwargs)`
A `FeatureFlowView` for loading a course share link which directly opens the course modal on the frontend. Therefore, this view overrides the `get_feature_flow` method to fill `intData` with the detailed course json for the modal.abs

Saves a `SharedCourseView` for analytics purposes.

get_feature_flow(`request, code, sem_name, year`)
Return data needed for the feature flow for this `HomeView`. A name value is automatically added in `.get()` using the `feature_name` class variable. A semester value can also be provided, which will change the initial semester state of the home page.

class `courses.views.SchoolList(**kwargs)`

get(`request, school`)
Provides the basic school information including the schools areas, departments, levels, and the time the data was last updated

`courses.views.all_courses(request)`
Generates the full course directory page. Includes links to all courses and is sorted by department.

`courses.views.course_page(request, code)`
Generates a static course page for the provided course code and school (via subdomain). Completely outside of the React framework purely via Django templates.

`courses.views.get_classmates_in_course(request, school, sem_name, year, course_id)`
 Finds all classmates for the authenticated user who also have a timetable with the given course.

Utils

`courses.utils.get_sections_by_section_type(course, semester)`
 Return a map from section type to Sections for a given course and semester.

`courses.utils.sections_are_filled(sections)`
 Return True if all sections are filled beyond their max enrollment.

Serializers

`class courses.serializers.CourseSerializer(*args, **kwargs)`
 Serialize a Course into a dictionary with detailed information about the course, and all related entities (eg Sections). Used for search results and course modals.

Takes a context with parameters: school: str (required) semester: Semester (required) student: Student (optional)

`get_evals(course)`
 Append all eval instances with a flag designating whether there exists another eval for the course with the same term+year values. :returns: List of modified evaluation dictionaries (added flag 'unique_term_year')

`get_popularity_percent(course)`
 Return percentage of course capacity that is filled by registered students.

`get_regexed_courses(course)`
 Given course data, search for all occurrences of a course code in the course description and prereq info and return a map from course code to course name for each course code.

`class courses.serializers.EvaluationSerializer(*args, **kwargs)`

`class courses.serializers.SectionSerializer(*args, **kwargs)`

`class courses.serializers.SemesterSerializer(*args, **kwargs)`

`courses.serializers.get_section_dict(section)`
 Returns a dictionary of a section including indicator of whether that section is filled

Student App

The Student model is an abstraction over the Django user to provide us with a more full user profile including information pulled from social authentication via Google and/or Facebook (and/or Microsoft JHED at JHU). This app handles utilities for overriding the Python Social Auth authentication pipeline, while also handling the functionality for logged in users.

The student app also encapsulates all models tied directly to a user like PersonalTimetables, PersonalEvents, Reactions, and notification tokens.

Models

Models pertaining to Students.

class `student.models.PersonalEvent(*args, **kwargs)`

A custom event that has been saved to a user's PersonalTimetable so that it persists across refresh, device, and session. Marks when a user is not free. Courses are scheduled around it.

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `student.models.PersonalTimetable(*args, **kwargs)`

Database object representing a timetable created (and saved) by a user.

A PersonalTimetable belongs to a Student, and contains a list of Courses and Sections that it represents.

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `student.models.Reaction(*args, **kwargs)`

Database object representing a reaction to a course.

A Reaction is performed by a Student on a Course, and can be one of REACTION_CHOICES below. The reaction itself is represented by its *title* field.

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `student.models.RegistrationToken(*args, **kwargs)`

A push notification token for Chrome notification via Google Cloud Messaging

exception `DoesNotExist`

exception `MultipleObjectsReturned`

class `student.models.Student(*args, **kwargs)`

Database object representing a student.

A student is the core user of the app. Thus, a student will have a class year, major, friends, etc. An object is only created for the user if they have signed up (that is, signed out users are not represented by Student objects).

exception `DoesNotExist`

exception `MultipleObjectsReturned`

Views

class `student.views.ClassmateView(**kwargs)`

Handles the computation of classmates for a given course, timetable, or simply the count of all classmates for a given timetable.

get(*request, sem_name, year*)

Returns

If the query parameter 'count' is present Information regarding the number of friends only:

```
{
  "id": Course with the most friends,
  "count": The maximum # of friends in a course,
  "total_count": the total # in all classes on timetable,
}
```

If the query parameter `course_ids` is present a list of dictionaries representing past classmates and current classmates. These are students who the authenticated user is friends with and who has social courses enabled.:

```
[{
  "course_id":6137,
  "past_classmates":[...],
  "classmates":[...]
}, ...]
```

Otherwise a list of friends and non-friends alike who have `social_all` enabled to be displayed in the “find-friends” modal. Sorted by the number courses the authenticated user shares.:

```
[{
  "name": "...",
  "is_friend": Whether or not the user is current user's friend,
  "profile_url": link to FB profile,
  "shared_courses": [...],
  "peer": Info about the user,
}, ...]
```

class `student.views.PersonalEventView(**kwargs)`

class `student.views.ReactionView(**kwargs)`

Manages the creation of Reactions to courses.

post(*request*)

Create a Reaction for the given course id, with the given title matching one of the possible emojis. If already present, remove that reaction.

class `student.views.UserTimetablePreferenceView(**kwargs)`

Used to update timetable preferences

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using `self.queryset`.

This method should always be used rather than accessing `self.queryset` directly, as `self.queryset` gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

serializer_class

alias of `timetable.serializers.PersonalTimeTablePreferencesSerializer`

class `student.views.UserTimetableView(**kwargs)`

Responsible for the viewing and managing of all Students' PersonalTimetable.

delete(*request, sem_name, year, tt_name*)

Deletes a PersonalTimetable by name/year/term.

get(*request, sem_name, year*)
Returns student's personal timetables

post(*request*)
Duplicates a personal timetable if a 'source' is provided. Else, creates a personal timetable based on the courses, custom events, preferences, etc. which are provided.

update_events(*tt, events*)
Replace tt's events with input events. Deletes all old events to avoid buildup in db

class `student.views.UserView(**kwargs)`
Handles the accessing and mutating of user information and preferences.

delete(*request*)
Delete this user and all of its data

get(*request*)
Renders the user profile/stats page which indicates all of a student's reviews of courses, what social they have connected, whether notificaitons are enabled, etc.

patch(*request*)
Updates a user settings to match the corresponding values passed in the request body. (e.g. `social_courses`, `class_year`, `major`)

`student.views.accept_tos(request)`
Accepts the terms of services for a user, saving the `datetime` the terms were accepted.

`student.views.log_ical_export(request)`
Logs that a calendar was exported on the frontend and indicates it was downloaded rather than exported to Google calendar.

Utils

`student.utils.get_classmates_from_course_id(school, student, course_id, semester, friends=None, include_same_as=False)`

Get's current and past classmates (students with timetables containing the provided course ID). Classmates must have `social_courses` enabled to be included. If `social_sections` is enabled, info about what section they are in is also passed.

Parameters

- **school** (`str`) – the school code (e.g. 'jhu')
- **student** (`Student`) – the student for whom to find classmates
- **course_id** (`int`) – the database id for the course
- **semester** (`Semester`) – the semester that is current (to check for)
- **friends** (`list` of `Students`) – if provided, does not re-query for friends list, uses provided list.
- **include_same_as** (`bool`) – If provided as true, searches for classmates in any courses marked as "same as" in the database.

`student.utils.get_classmates_from_tts(student, course_id, tts)`
Returns a list of classmates a student has from a list of other user's timetables. This utility does the leg work for `get_classmates_from_course_id()` by taking either a list of current or past timetables and finding classmates relevant to that list.

If both students have `social_offerings` enabled, adds information about what sections the student is enrolled in on each classmate.

`student.utils.get_friend_count_from_course_id(student, course_id, semester)`

Computes the number of friends a user has in a given course for a given semester.

Ignores whether or not those friends have social courses enabled. Never exposes those user's names or information. This count is used purely to upsell user's to enable social courses.

`student.utils.get_student(request)`

Returns the student belonging to the authenticated user

Return type (Student)

`student.utils.get_student_tts(student, school, semester)`

Returns serialized list of a student's `PersonalTimetable` objects ordered by last updated for passing to the frontend.

Serializers

`class student.serializers.StudentSerializer(*args, **kwargs)`

`student.serializers.get_student_dict(school, student, semester)`

Return serialized representation of a student.

Searches App

Searches app provides a useful, efficient and scalable search backend using basic techniques of information retrieval.

Views

`class searches.views.CourseSearchList(**kwargs)`

Course Search List.

`get(request, query, sem_name, year)`

Return search results.

`post(request, query, sem_name, year)`

Return advanced search results.

Utils

`searches.utils.course_name_contains_token(token)`

Returns a query set of courses where tokens are contained in code or name.

`searches.utils.search(school, query, semester)`

Returns courses that are contained in the name from a query.

Agreement App

In order to use our system, users must agree to our privacy policy/terms and conditions.

When a user is not logged in, this is done implicitly (no click to accept is required). Out of respect for our users and to be fully transparent, we surface a banner when the user is not logged in to bring this implicitly agreement to their attention.

When a user is logged in, the agreement must be explicitly. During signup the user is prompted to agree to the terms and must do so in order to continue using the application. If the documents have been updated since the user last agreed, they will be notified of this change and once again asked to agree to the updated terms/policy.

Models

```
class agreement.models.Agreement(*args, **kwargs)
    Database object representing updates to the ToS/privacy policy.
```

```
    exception DoesNotExist
```

```
    exception MultipleObjectsReturned
```

E2E Test Utils

```
class semesterly.test_utils.SeleniumTestCase(*args, **kwargs)
```

This test case extends the Django StaticLiveServerTestCase. It creates a selenium ChromeDriver instance on setUp of each test. It navigates to the live url for the static live server. It also provides utilities and assertions for navigating and testing presence of elements or behavior.

```
    img_dir
```

Directory to save screenshots on failure.

```
        Type str
```

```
    driver
```

Chrome WebDriver instance.

```
        Type WebDriver
```

```
    timeout
```

Socket default timeout.

```
        Type int
```

```
    add_course(course_idx, n_slots, n_master_slots, by_section="", code=None)
```

Adds a course via search results and asserts the corresponding number of slots are found

Parameters

- **course_idx** (*int*) – index into the search results corresponding the to course to add
- **n_slots** (*int*) – the number of slots expected after add
- **n_master_slots** (*int*) – the number of master slots expected after add
- **by_section** (*str, optional*) – if provided adds the specific section of the course
- **code** (*str, optional*) – the course code to add, validates presence if provided

```
    add_course_from_course_modal(n_slots, n_master_slots)
```

Adds a course via the course modal action. Requires that the course modal be open.

allow_conflicts_add(*n_slots*)

Allows conflicts via the conflict alert action, then validates that the course was added

assert_custom_event_exists(*, *name: str*, *day: Optional[str] = None*, *location: Optional[str] = None*, *color: Optional[str] = None*, *start_time: Optional[str] = None*, *end_time: Optional[str] = None*, *credits: Optional[float] = None*)

Asserts that a custom event with the provided fields exists in the current timetable.

Parameters

- **name** – Name of the event, can be substring of the actual name
- **day** – Day of the week, one of “M”, “T”, “W”, “R”, “F”, “S”, “U”
- **location** – Location of the event, can be substring of the actual name
- **color** – Color of the event in hex (#F8F6F7), case insensitive
- **start_time** – Start time of the event as a non zero-padded string (8:00)
- **end_time** – End time of the event as a non zero-padded string (14:30)
- **credits** – Number of credits of the event

Raises **RuntimeError** – If the event could not be found.

assert_friend_image_found(*friend*)

Asserts that the provided friend’s image is found on the page

assert_friend_in_modal(*friend*)

Asserts that the provided friend’s image is found on the modal

assert_invisibility(*locator*, *root=None*)

Asserts the invisibility of the provided element

Parameters

- **locator** – A tuple of (By.*, ‘identifier’)
- **root** (*bool*, *optional*) – The root element to search from, root of DOM if None

assert_loader_completes()

Asserts that the semester.ly page loader has completed

assert_n_elements_found(*locator*, *n_elements*, *root=None*)

Asserts that *n_elements* are found by the provided locator

assert_ptt_const_across_refresh()

Refreshes the browser and asserts that the tuple version of the personal timetable is equivalent to pre-refresh

assert_ptt_equals(*ptt*)

Asserts equivalency between the provided ptt tuple and the current ptt

assert_slot_presence(*n_slots*, *n_master_slots*)

Assert *n_slots* and *n_master_slots* are on the page

change_ptt_name(*name*)

Changes personal timetable name to the provided title

change_term(*term*, *clear_alert=False*)

Changes the term to the provided term by matching the string to the string found in the semester dropdown on Semester.ly

clear_search_query()

Clears the search box

clear_tutorial()

Clears the tutorial modal for first time users

click_off()

Clears the focus of the driver

close_adv_search()

Closes the advanced search modal

close_course_modal()

Closes the course modal using the (x) button

compare_timetable(*timetable_name: str*)

Activates the compare timetable mode with a timetable of the given name.

Parameters **timetable_name** – Name of the timetable to compare to, must already exist.

Pre-condition: The timetable dropdown is not clicked.

complete_user_settings_basics(*major, class_year*)

Completes major/class year/TOS agreement via the welcome modal

Parameters

- **major** (*str*) – Student's major
- **class_year** (*str*) – Student's class year

create_custom_event(*day: int, start_time: int, end_time: int, show_weekend: bool = True*)

Creates a custom event using drag and drop assuming custom event mode is off

Parameters

- **day** – 0-6, 0 is Monday
- **start_time** – 0 is 8:00A.M, every 1 is 30 mins
- **end_time** – 0 is 8:00A.M, every 1 is 30 mins
- **show_weekend** – if weekends are shown

create_friend(*first_name, last_name, **kwargs*)

Creates a friend of the primary (first) user

create_personal_timetable_obj(*friend, courses, semester*)

Creates a personal timetable object belonging to the provided user with the given courses and semester

create_ptt(*name: str = "", finish_saving: bool = True*)

Create a personaltimetable with the provided name when provided

Parameters

- **name** – Name of the personal timetable
- **finish_saving** – Whether to wait until the personal timetable is saved

description(*descr*)

A context manager which wraps a group of code and adds details to any exceptions thrown by the enclosed lines. Upon such an exception, the context manager will also take a screenshot of the current state of self.driver, writing a PNG to self.img_dir, labeled by the provided description and a timestamp.

edit_custom_event(*old_name: str, /, *, name: Optional[str] = None, day: Optional[str] = None, location: Optional[str] = None, color: Optional[str] = None, start_time: Optional[str] = None, end_time: Optional[str] = None, credits: Optional[float] = None*)

Edits the first custom event found with the provided name.

Parameters

- **old_name** – The name of the event to edit.
- **name** – The new name to give the event.
- **day** – The new day of the week, one of “M”, “T”, “W”, “R”, “F”, “S”, “U”.
- **location** – The new location.
- **color** – The new color as a hex code (#FF0000).
- **start_time** – The new start time in military time (8:00).
- **end_time** – The new end time in military time (13:00).
- **credits** – The new number of credits.

enter_search_query(*query*)

Enters the provided query into the search box

execute_action_expect_alert(*action, alert_text_contains=""*)

Executes the provided action, asserts that an alert appears and validates that the alert text contains the provided string (when provided)

exit_compare_timetable()

Exits the compare timetable mode (pre: already in compare timetable mode)

find(*locator, get_all=False, root=None, clickable=False, hidden=False*) → WebElement | list[WebElement]

Locates element in the DOM and returns it when found.

Parameters

- **locator** – A tuple of (By.*, ‘identifier’)
- **get_all** (*bool, optional*) – If true, will return list of matching elements
- **root** (*bool, optional*) – The root element to search from, root of DOM if None
- **clickable** (*bool, optional*) – If true, waits for clickability of element
- **hidden** (*bool, optional*) – If true, will allow for hidden elements

Returns The WebElement object returned by self.driver (Selenium)

Throws: RuntimeError: If element is not found or both get_all and clickable is True

follow_and_validate_url(*url, validate*)

Opens a new window, switches to it, gets the url and validates it using the provided validating function.

Parameters

- **url** (*str*) – the url to follow and validate
- **validate** (*func*) – the function which validates the new page

follow_share_link_from_slot()

Click the share link on the slot and follow it then validate the course modal

get_custom_event_fields()

Returns the fields of the currently selected custom event.

Pre-condition: Custom event modal is open.

get_elements_as_text(*locator*)

Gets elements using self.get and represents them as text

get_test_url(*school*, *path=""*)

Get's the live server testing url for a given school.

Parameters

- **school** (*str*) – the string for which to create the test url
- **path** (*str*) – the appended path to file or page with trailing /

Returns the testing url

get_timetable_name()

Gets the personal timetable name

init_screenshot_dir()

Initializes directory to which we store test failure screenshots

lock_course()

Locks the first course on the timetable

login_via_fb(*email*, *password*)

Login user via fb by detecting the Continue with Facebook button in the signup modal, and then mocking user's credentials

Parameters

- **email** (*str*) – User's email
- **password** (*str*) – User's password

login_via_google(*email*, *password*)

Mocks the login of a user via Google by detecting the Continue with Google button in the signup modal, and then mocking the user's credentials.

Parameters

- **email** (*str*) – User's email
- **password** (*str*) – User's password

open_and_query_adv_search(*query*, *n_results=None*)

Open's the advanced search modal and types in the provided query, asserting that *n_results* are then returned

open_course_modal_from_search(*course_idx*)

Opens course modal from search by search result index

open_course_modal_from_slot(*course_idx*)

Opens the course modal from the *n*th slot

ptt_to_tuple()

Converts personal timetable to a tuple representation

remove_course(*course_idx*, *from_slot=False*, *n_slots_expected=None*)

Removes a course from the user's timetable, asserts master slot is removed.

Parameters

- **course_idx** (*int*) – the index of the course for which to remove
- **from_slot** (*bool*, *optional*) – if provided, removes via slot rather than via a master_slot
- **n_slots_expected** (*int*, *optional*) – if provided, asserts *n* slots found after removal

remove_course_from_course_modal(*n_slots_expected=None*)

Removes course via the action within the course's course modal. Requires that the course modal be open.

save_user_settings()

Saves user settings by clicking the button, asserts that the modal is then invisible

search_course(*query*, *n_results*)

Searches a course and asserts *n_results* elements are found

Parameters

- **query** (*str*) – the text to enter into search
- **n_results** (*int*) – the number of results to look for. If 0, will look for no results

select_nth_adv_search_result(*index*, *semester*)

Selects the *nth* advanced search result with a click. Validates the course modal body displayed in the search results

setUp()

Hook method for setting up the test fixture before exercising it.

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

share_timetable(*courses*)

Clicks the share button via the top bar and validates it. Validation is done by following the url and checking the timetable using the `validate_timetable` function

switch_to_ptt(*name*)

Switches to the personal timetable with matching name

take_alert_action()

Takes the action provided by the alert by clicking the button on when visible

tearDown()

Hook method for deconstructing the test fixture after testing it.

validate_course_modal()

Validates the course modal displays proper course data

validate_course_modal_body(*course*, *modal*, *semester*)

Validates the course modal body displays credits, name, code, etc.

validate_timeable(*courses*)

Validate timetable by checking that for each course provided, a slot exists with that course's name and course code.

semesterly.test_utils.force_login(*user*, *driver*, *base_url*)

Forces the login of the provided user setting all cookies. Function will refresh the provided driver and the user will be logged in to that session.

class semesterly.test_utils.function_returns_true(*func*)

An expectation for checking if the provided function returns true

class semesterly.test_utils.n_elements_to_be_found(*locator*, *n_*)

An expectation for checking if the *n* elements are found locator, text

class semesterly.test_utils.text_to_be_present_in_element_attribute(*locator*, *text_*, *attribute_*)

An expectation for checking if the given text is present in the element's locator, text

class semesterly.test_utils.text_to_be_present_in_nth_element(*locator*, *text_*, *index_*)

An expectation for checking if the given text is present in the *nth* element's locator, text

class semesterly.test_utils.url_matches_regex(*pattern*)

Expected Condition which waits until the browser's url matches the provided regex

Helpers App

Decorators

`helpers.decorators.validate_subdomain(view_func)`
Validates subdomain, redirecting user to index if the school is invalid.

Mixins

`class helpers.mixins.CsrfExemptSessionAuthentication`

`enforce_csrf(request)`
Enforce CSRF validation for session based authentication.

`class helpers.mixins.FeatureFlowView(**kwargs)`
Template that handles GET requests by rendering the homepage. `Feature_name` or `get_feature_flow()` can be overridden to launch a feature or action on homepage load.

`get_feature_flow(request, *args, **kwargs)`
Return data needed for the feature flow for this HomeView. A `name` value is automatically added in `.get()` using the `feature_name` class variable. A `semester` value can also be provided, which will change the initial semester state of the home page.

`class helpers.mixins.RedirectToSignupMixin`

`class helpers.mixins.ValidateSubdomainMixin`
Mixin which validates subdomain, redirecting user to index if the school is not in `ACTIVE_SCHOOLS`.

Authentication Pipeline

Views

`class authpipe.views.RegistrationTokenView(**kwargs)`
Handles registration and deletion of tokens for maintaining chrome notifications for users who choose to enable the feature.

`put(request)`
Creates a notification token for the user.

Utils

`authpipe.utils.associate_students(strategy, details, response, user, *args, **kwargs)`
Part of our custom Python Social Auth authentication pipeline. If a user already has an account associated with an email, associates that user with the new backend.

`authpipe.utils.check_student_token(student, token)`
Validates a token: checks that it is at most 2 days old and that it matches the currently authenticated student.

`authpipe.utils.create_student(strategy, details, response, user, *args, **kwargs)`
Part of the Python Social Auth pipeline which creates a student upon signup. If student already exists, updates information from Facebook or Google (depending on the backend). Saves friends and other information to fill database.

1.3.11 Flows Documentation

Initialization

When a user loads the home timetable page, `FeatureFlowView` inside of `timetable.utils` is used to handle the request. On initial page load, the frontend requires some data to initialize the redux state, like information about the current user, the list of possible semesters for the school, and the list of student integrations. This initial data is created inside of the view, and passed in as a single json string in the response context:

```
class FeatureFlowView(ValidateSubdomainMixin, APIView):

    def get(self, request, *args, **kwargs):
        # ...gather values for init_data

        init_data = {
            'school': self.school,
            'currentUser': get_user_dict(self.school, self.student, sem),
            'currentSemester': curr_sem_index,
            'allSemesters': all_semesters,
            'uses12HrTime': self.school in AM_PM_SCHOOLS,
            'studentIntegrations': integrations,
            'examSupportedSemesters': map(all_semesters.index,
                                        final_exams_available.get(self.
↪school, [])),

            'featureFlow': dict(feature_flow, name=self.feature_name)
        }

        return render(request, 'timetable.html', {'init_data': json.dumps(init_
↪data)})
```

which makes the `init_data` variable accessible in `timetable.html`. This dumped json string is then passed to the frontend as a global variable:

```
<script type="text/javascript">
  var initData = "{{init_data|escapejs}}";
</script>
```

And then parsed inside of the `setup()` function in `init.jsx`

```
const setup = () => (dispatch) => {
  initData = JSON.parse(initData);

  // pass init data into the redux state
  dispatch({ type: ActionTypes.INIT_STATE, data: initData });

  // do other logic with initData...
};
```

In other words, the data that the frontend requires is retrieved/calculated inside of `FeatureFlowView`, and then passed to the frontend as global variable `initData`. The frontend then does any logic it needs based on that data inside of `setup()` in `init.jsx`. Any data that needs to be reused later on from `initData` should be passed in to the redux state so that the only global variable uses appear in `setup()`.

Feature Flows

One such piece of data that is passed to the frontend is a `featureFlow` object. This object is obtained as the return value of `.get_feature_flow()`, in addition to a `name: self.feature_name` key value pair. In the default implementation, this is just the dictionary `{name: None}`:

```
class FeatureFlowView(ValidateSubdomainMixin, APIView):
    feature_name = None

    def get_feature_flow(self, request, *args, **kwargs):
        return {}

    def get(self, request, *args, **kwargs):
        ...
        feature_flow = self.get_feature_flow(request, *args, **kwargs)
        init_data = {
            ...
            'featureFlow': dict(feature_flow, name=self.feature_name)
        }

        return render(request, 'timetable.html', {'init_data': json.dumps(init_
        ↪data)})
```

This feature flow value can be used to store any extra information that the frontend needs for any endpoints that would require initial data to be loaded. For example, when loading a timetable share link, the frontend also needs to get data about the timetable that is being shared - instead of making a request to the backend after page load, this information can be provided by the backend directly by passing this information in the feature flow. It is easy to write new views that pass different data and have custom logic by subclassing `FeatureFlowView` and overwriting the `get_feature_flow()` method and the `.feature_name` class attribute.

Having this data all stored under the key `featureFlow` in `init_data` ensures two things. Firstly, it makes explicit that there can only be one feature flow in play at a time (we can't load a timetable share link and a course share link at the same time), and secondly, it allows the frontend to know where to look for any feature data and act accordingly. In practice, this is done by switching on the name of the feature flow:

```
const setup = () => (dispatch) => {
    initData = JSON.parse(initData);

    dispatch({ type: ActionTypes.INIT_STATE, data: initData });

    // do other logic with initData...

    dispatch(handleFlows(initData.featureFlow));
};

const handleFlows = featureFlow => (dispatch) => {
    switch (featureFlow.name) {
        case 'SIGNUP':
            dispatch(signupModalActions.showSignupModal());
            break;
        case 'USER_ACQ':
            dispatch(userAcquisitionModalActions.triggerAcquisitionModal());
            break;
        case 'SHARE_TIMETABLE':
```

(continues on next page)

(continued from previous page)

```

        dispatch(timetablesActions.cachedTimetableLoaded());
        dispatch(lockTimetable(featureFlow.sharedTimetable, true, initData.
↪currentUser.isLoggedIn));
        break;
        // ... etc.
        default:
            // unexpected feature name
            break;
    }
};

```

Example

To help understand how feature flows work, let's go through the code for an example feature flow: course sharing. In order to implement course sharing, we want to create a new view/endpoint that retrieves course data based on the url and passes it to the frontend, which would then update the redux state and dispatch an action to open the course modal.

We start by defining a new endpoint for this feature flow:

```

re_path(r'course/(?P<code>.+)/(?P<sem_name>.+)/(?P<year>.+)*$',
        courses.views.CourseModal.as_view())

```

Then we create a new `FeatureFlowView` for this endpoint which needs to do two things: define a name for the feature flow, which the frontend look at to determine what action to do, and return the course data that the frontend needs inside of `get_feature_flow()`:

```

class CourseModal(FeatureFlowView):
    feature_name = "SHARE_COURSE"

    def get_feature_flow(self, request, code, sem_name, year):
        semester, _ = Semester.objects.get_or_create(name=sem_name, year=year)
        code = code.upper()
        course = get_object_or_404(Course, school=self.school, code=code)
        course_json = get_detailed_course_json(self.school, course, semester, ↪
↪self.student)

        # analytics
        SharedCourseView.objects.create(
            student=self.student,
            shared_course=course,
        ).save()

        return {'sharedCourse': course_json, 'semester': semester}

```

The frontend can now add a new case in `handleFlows` to perform logic for this feature flow:

```

const handleFlows = featureFlow => (dispatch) => {
    switch (featureFlow.name) {
        ...
        case 'SHARE_COURSE':
            dispatch(setCourseInfo(featureFlow.sharedCourse));
            dispatch(fetchCourseClassmates(featureFlow.sharedCourse.id));

```

(continues on next page)

(continued from previous page)

```
        break;
    // ... etc.
    default:
        // unexpected feature name
        break;
    }
};
```

Shortcuts

Some feature flows don't require any extra data - they simply require the frontend to know that a feature flow is being run. For example, for the signup feature flow, loading the page at `/signup` should simply open the signup modal, which requires no extra logic or data other than knowing that it should occur. We could do this by writing a new view:

```
class SignupModal(FeatureFlowView):
    feature_name = "SIGNUP"
```

We do not need to implement `.get_feature_flow()` since the frontend doesn't require any extra data and the default implementation already returns an empty dictionary. We can simplify this by simply declaring this view directly inside of the urls file:

```
re_path(r'^signup/*$', FeatureFlowView.as_view(feature_name='SIGNUP'))
```

see <https://github.com/noahpresler/semesterly/pull/838> for the original pull request implementing feature flows

1.3.12 Data Pipeline Documentation

Semester.ly's data pipeline provides the infrastructure by which the database is filled with course information. Whether a given University offers an API or an online course catalogue, this pipeline lends developers an easy framework to work within to pull that information and save it in our Django Model format.

General System Workflow

1. Pull HTML/JSON markup from a catalogue/API
2. Map the fields of the mark up to the fields of our ingestor (by simply filling a python dictionary).
3. The ingestor preprocesses the data, validates it, and writes it to JSON.
4. Load the JSON into the database.

Note: This process happens automatically via [Django/Celery Beat Periodict Tasks](#). You can learn more about these schedule tasks below ([Scheduled Tasks](#)).

Steps 1 and 2 are what we call **parsing** – an operation that is non-generalizable across all Universities. Often a new parser must be written. For more information on this, read [Add a School](#).

Parsing Library Documentation

Base Parser

Requester

Ingestor

exception `parsing.library.ingestor.IngestionError`(*data*, **args*)

Bases: `parsing.library.exceptions.PipelineError`

Ingestor error class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `parsing.library.ingestor.IngestionWarning`(*data*, **args*)

Bases: `parsing.library.exceptions.PipelineWarning`

Ingestor warning class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `parsing.library.ingestor.Ingestor`(*config*, *output*, *break_on_error=True*, *break_on_warning=False*, *display_progress_bar=True*, *skip_duplicates=True*, *validate=True*, *tracker=<parsing.library.tracker.NullTracker object>*)

Bases: `dict`

Ingest parsing data into formatted json.

Mimics functionality of dict.

ALL_KEYS

Set of keys supported by Ingestor.

Type `set`

break_on_error

Break/cont on errors.

Type `bool`

break_on_warning

Break/cont on warnings.

Type `bool`

school

School code (e.g. jhu, gw, umich).

Type `str`

skip_duplicates

Skip ingestion for repeated definitions.

Type `bool`

tracker

Tracker object.

Type library.tracker

UNICODE_WHITESPACE

regex that matches Unicode whitespace.

Type TYPE

validate

Enable/disable validation.

Type bool

validator

Validator instance.

Type library.validator

```
ALL_KEYS = {'areas', 'campus', 'capacity', 'code', 'coreqs', 'corequisites',
'cores', 'cost', 'course', 'course_code', 'course_name', 'course_section_id',
'credits', 'date', 'date_end', 'date_start', 'dates', 'day', 'days', 'department',
'department_code', 'department_name', 'dept_code', 'dept_name', 'descr',
'description', 'end_time', 'enrollment', 'enrolment', 'exclusions', 'fee', 'fees',
'final_exam', 'geneds', 'homepage', 'instr', 'instr_name', 'instr_names', 'instrs',
'instructor', 'instructor_name', 'instructors', 'kind', 'level', 'loc', 'location',
'meeting_section', 'meetings', 'name', 'num_credits', 'offerings', 'pos', 'prereqs',
'prerequisites', 'remaining_seats', 'same_as', 'school', 'school_subdivision_code',
'school_subdivision_name', 'score', 'section', 'section_code', 'section_name',
'section_type', 'sections', 'semester', 'size', 'start_time', 'sub_school',
'summary', 'term', 'time', 'time_end', 'time_start', 'type', 'waitlist',
'waitlist_size', 'website', 'where', 'writing_intensive', 'year'}
```

clear() → None. Remove all items from D.

copy() → a shallow copy of D

end()

Finish ingesting.

Close i/o, clear internal state, write meta info

fromkeys(value=None, /)

Create a new dictionary with keys from iterable and values set to value.

get(key, default=None, /)

Return the value for key if key is in the dictionary, else default.

ingest_course()

Create course json from info in model map.

Returns course

Return type dict

ingest_eval()

Create evaluation json object.

Returns eval

Return type dict

ingest_meeting(*section*, *clean_only=False*)

Create meeting ingested json map.

Parameters **section** (*dict*) – validated section object

Returns meeting

Return type *dict*

ingest_section(*course*)

Create section json object from info in model map.

Parameters **course** (*dict*) – validated course object

Returns section

Return type *dict*

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*, [*d*]) → *v*, remove specified key and return the corresponding value.

If key is not found, *d* is returned if given, otherwise `KeyError` is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises `KeyError` if the dict is empty.

setdefault(*key*, *default=None*, /)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update([*E*], ***F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for *k* in E: `D[k] = E[k]` If E is present and lacks a `.keys()` method, then does: for *k*, *v* in E: `D[k] = v` In either case, this is followed by: for *k* in F: `D[k] = F[k]`

values() → an object providing a view on D's values

Validator

exception `parsing.library.validator.MultipleDefinitionsWarning`(*data*, **args*)

Bases: `parsing.library.validator.ValidationWarning`

Duplicated key in data definition.

args

with_traceback()

Exception.with_traceback(*tb*) – set self.__traceback__ to *tb* and return self.

exception `parsing.library.validator.ValidationError`(*data*, **args*)

Bases: `parsing.library.exceptions.PipelineError`

Validator error class.

args

with_traceback()

Exception.with_traceback(*tb*) – set self.__traceback__ to *tb* and return self.

exception `parsing.library.validator.ValidationWarning`(*data*, **args*)

Bases: `parsing.library.exceptions.PipelineWarning`

Validator warning class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `parsing.library.validator.Validator`(*config*, *tracker=None*, *relative=True*)

Bases: `object`

Validation engine in parsing data pipeline.

config

Loaded config.json.

Type `DotDict`

course_code_regex

Regex to match course code.

Type `re`

kind_to_validation_function

Map kind to validation function defined within this class.

Type `dict`

KINDS

Kinds of objects that validator validates.

Type `set`

relative

Enforce relative ordering in validation.

Type `bool`

seen

Running monitor of seen courses and sections

Type `dict`

tracker

Type `parsing.library.tracker.Tracker`

KINDS = {'config', 'course', 'datalist', 'directory', 'eval', 'final_exam', 'instructor', 'meeting', 'section'}

static `file_to_json`(*path*, *allow_duplicates=False*)

Load file pointed to by path into json object dictionary.

Parameters

- **path** (*str*) –
- **allow_duplicates** (*bool*, *optional*) – Allow duplicate keys in JSON.

Returns JSON-compliant dictionary.

Return type `dict`

classmethod `load_schemas`(*schema_path=None*)

Load JSON validation schemas.

NOTE: Will load schemas as static variable (i.e. once per definition), unless `schema_path` is specifically defined.

Parameters `schema_path` (*None, str, optional*) – Override default `schema_path`

static `schema_validate`(*data, schema, resolver=None*)

Validate data object with JSON schema alone.

Parameters

- **data** (*dict*) – Data object to validate.
- **schema** – JSON schema to validate against.
- **resolver** (*None, optional*) – JSON Schema reference resolution.

Raises `jsonschema.exceptions.ValidationError` – Invalid object.

validate(*data, transact=True*)

Validation entry/dispatcher.

Parameters `data` (*list, dict*) – Data to validate.

validate_course(*course*)

Validate course.

Parameters `course` (*DotDict*) – Course object to validate.

Raises

- ***MultipleDefinitionsWarning*** – Course has already been validated in same session.
- ***ValidationError*** – Invalid course.

validate_directory(*directory*)

Validate directory.

Parameters `directory` (*str, dict*) – Directory to validate. May be either path or object.

Raises ***ValidationError*** – encapsulated `IOError`

validate_eval(*course_eval*)

Validate evaluation object.

Parameters `course_eval` (*DotDict*) – Evaluation to validate.

Raises ***ValidationError*** – Invalid evaluation.

validate_final_exam(*final_exam*)

Validate final exam.

NOTE: currently unused.

Parameters `final_exam` (*DotDict*) – Final Exam object to validate.

Raises ***ValidationError*** – Invalid final exam.

validate_instructor(*instructor*)

Validate instructor object.

Parameters `instructor` (*DotDict*) – Instructor object to validate.

Raises ***ValidationError*** – Invalid instructor.

validate_location(*location*)

Validate location.

Parameters `location` (`DotDict`) – Location object to validate.

Raises `ValidationWarning` – Invalid location.

validate_meeting(*meeting*)

Validate meeting object.

Parameters `meeting` (`DotDict`) – Meeting object to validate.

Raises

- `ValidationError` – Invalid meeting.
- `ValidationWarning` – Description

validate_section(*section*)

Validate section object.

Parameters `section` (`DotDict`) – Section object to validate.

Raises

- `MultipleDefinitionsWarning` – Invalid section.
- `ValidationError` – Description

validate_self_contained(*data_path*, *break_on_error=True*, *break_on_warning=False*,
output_error=None, *display_progress_bar=True*, *master_log_path=None*)

Validate JSON file as without ingestor.

Parameters

- `data_path` (*str*) – Path to data file.
- `break_on_error` (*bool*, *optional*) – Description
- `break_on_warning` (*bool*, *optional*) – Description
- `output_error` (*None*, *optional*) – Error output file path.
- `display_progress_bar` (*bool*, *optional*) – Description
- `master_log_path` (*None*, *optional*) – Description
- `break_on_error` –
- `break_on_warning` –
- `display_progress_bar` –

Raises `ValidationError` – Description

validate_time_range(*start*, *end*)

Validate start time and end time.

There exists an unhandled case if the end time is midnight.

Parameters

- `start` (*str*) – Start time.
- `end` (*str*) – End time.

Raises `ValidationError` – Time range is invalid.

static validate_website(*url*)

Validate url by sending HEAD request and analyzing response.

Parameters `url` (*str*) – URL to validate.

Raises `ValidationError` – URL is invalid.

Logger

```
class parsing.library.logger.JSONColoredFormatter(fmt=None, datefmt=None, style='%',
                                                validate=True)
```

Bases: `logging.Formatter`

converter()

```
localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
                        tm_sec,tm_wday,tm_yday,tm_isdst)
```

Convert seconds since the Epoch to a time tuple expressing local time. When ‘seconds’ is not passed in, convert the current time instead.

```
default_msec_format = '%s,%03d'
```

```
default_time_format = '%Y-%m-%d %H:%M:%S'
```

format(record)

Format the specified record as text.

The record’s attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using `LogRecord.getMessage()`. If the formatting string uses the time (as determined by a call to `usesTime()`), `formatTime()` is called to format the event time. If there is exception information, it is formatted using `formatException()` and appended to the message.

formatException(ei)

Format and return the specified exception information as a string.

This default implementation just uses `traceback.print_exception()`

formatMessage(record)

formatStack(stack_info)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

formatTime(record, datefmt=None)

Return the creation time of the specified `LogRecord` as formatted text.

This method should be called from `format()` by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if `datefmt` (a string) is specified, it is used with `time.strftime()` to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, `time.localtime()` is used; to change this for a particular formatter instance, set the ‘converter’ attribute to a function with the same signature as `time.localtime()` or `time.gmtime()`. To change it for all formatters, for example if you want all logging times to be shown in GMT, set the ‘converter’ attribute in the `Formatter` class.

usesTime()

Check if the format uses the creation time of the record.

class `parsing.library.logger.JSONFormatter`(*fmt=None, datefmt=None, style='%', validate=True*)

Bases: `logging.Formatter`

Simple JSON extension of Python `logging.Formatter`.

converter()

localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

default_msec_format = '%s,%03d'

default_time_format = '%Y-%m-%d %H:%M:%S'

format(*record*)

Format record message.

Parameters *record* (`logging.LogRecord`) – Description

Returns Prettified JSON string.

Return type `str`

formatException(*ei*)

Format and return the specified exception information as a string.

This default implementation just uses `traceback.print_exception()`

formatMessage(*record*)

formatStack(*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

formatTime(*record, datefmt=None*)

Return the creation time of the specified `LogRecord` as formatted text.

This method should be called from `format()` by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if `datefmt` (a string) is specified, it is used with `time.strftime()` to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, `time.localtime()` is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as `time.localtime()` or `time.gmtime()`. To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the `Formatter` class.

usesTime()

Check if the format uses the creation time of the record.

class `parsing.library.logger.JSONStreamWriter`(*obj, type_=<class 'list'>, level=0*)

Bases: `object`

Context to stream JSON list to file.

BRACES

Open close brace definitions.

Type TYPE

file

Current object being JSONified and streamed.

Type dict

first

Indicator if first write has been done by streamer.

Type bool

level

Nesting level of streamer.

Type int

type_

Actual type class of streamer (dict or list).

Type dict, list

Examples

```
>>> with JSONStreamWriter(sys.stdout, type_=dict) as streamer:
...     streamer.write('a', 1)
...     streamer.write('b', 2)
...     streamer.write('c', 3)
{
  "a": 1,
  "b": 2,
  "c": 3
}
>>> with JSONStreamWriter(sys.stdout, type_=dict) as streamer:
...     streamer.write('a', 1)
...     with streamer.write('data', type_=list) as streamer2:
...         streamer2.write({0:0, 1:1, 2:2})
...         streamer2.write({3:3, 4:'4'})
...     streamer.write('b', 2)
{
  "a": 1,
  "data":
  [
    {
      0: 0,
      1: 1,
      2: 2
    },
    {
      3: 3,
      4: "4"
    }
  ],
  "b": 2
}
```

```
BRACES = {<class 'list'>: ('[', ']'), <class 'dict'>: ('{', ')')}
```

enter()

Wrapper for self.__enter__.

exit()

Wrapper for self.__exit__.

write(*args, **kwargs)

Write to JSON in streaming fasion.

Picks either write_obj or write_key_value

Parameters

- ***args** – pass-through
- ****kwargs** – pass-through

Returns return value of appropriate write function.

Raises **ValueError** – **type_** is not of type list or dict.

write_key_value(key, value=None, type_=<class 'list'>)

Write key, value pair as string to file.

If value is not given, returns new list streamer.

Parameters

- **key** (*str*) – Description
- **value** (*str, dict, None, optional*) – Description
- **type** (*str, optional*) – Description

Returns None if value is given, else new JSONStreamWriter

write_obj(obj)

Write obj as JSON to file.

Parameters **obj** (*dict*) – Serializable obj to write to file.

`parsing.library.logger.colored_json(j)`

Tracker

class `parsing.library.tracker.NullTracker(*args, **kwargs)`

Bases: `parsing.library.tracker.Tracker`

Dummy tracker used as an interface placeholder.

BROADCAST_TYPES = {'DEPARTMENT', 'INSTRUCTOR', 'MODE', 'SCHOOL', 'STATS', 'TERM', 'TIME', 'YEAR'}

add_viewer(viewer, name=None)

Add viewer to broadcast queue.

Parameters

- **viewer** (*Viewer*) – Viewer to add.
- **name** (*None, str, optional*) – Name the viewer.

broadcast(broadcast_type)

Do nothing.

property `department`

end()

End tracker and report to viewers.

get_viewer(*name*)

Get viewer by name.

Will return arbitrary match if multiple viewers with same name exist.

Parameters **name** (*str*) – Viewer name to get.

Returns Viewer instance if found, else None

Return type *Viewer*

has_viewer(*name*)

Determine if name exists in viewers.

Parameters **name** (*str*) – The name to check against.

Returns True if name in viewers else False

Return type *bool*

property instructor

property mode

remove_viewer(*name*)

Remove all viewers that match name.

Parameters **name** (*str*) – Viewer name to remove.

report()

Do nothing.

property school

start()

Start timer of tracker object.

property stats

property term

property time

property year

class `parsing.library.tracker.Tracker`

Bases: *object*

Tracks specified attributes and broadcasts to viewers.

@property attributes are defined for all BROADCAST_TYPES

```
BROADCAST_TYPES = {'DEPARTMENT', 'INSTRUCTOR', 'MODE', 'SCHOOL', 'STATS', 'TERM',
                   'TIME', 'YEAR'}
```

add_viewer(*viewer*, *name=None*)

Add viewer to broadcast queue.

Parameters

- **viewer** (*Viewer*) – Viewer to add.
- **name** (*None*, *str*, *optional*) – Name the viewer.

broadcast(*broadcast_type*)

Broadcast tracker update to viewers.

Parameters **broadcast_type** (*str*) – message to go along broadcast bus.

Raises *TrackerError* – if broadcast_type is not in BROADCAST_TYPE.

end()

End tracker and report to viewers.

get_viewer(*name*)

Get viewer by name.

Will return arbitrary match if multiple viewers with same name exist.

Parameters **name** (*str*) – Viewer name to get.

Returns Viewer instance if found, else None

Return type *Viewer*

has_viewer(*name*)

Determine if name exists in viewers.

Parameters **name** (*str*) – The name to check against.

Returns True if name in viewers else False

Return type *bool*

remove_viewer(*name*)

Remove all viewers that match name.

Parameters **name** (*str*) – Viewer name to remove.

report()

Notify viewers that tracker has ended.

start()

Start timer of tracker object.

exception `parsing.library.tracker.TrackerError`(*data*, **args*)

Bases: `parsing.library.exceptions.PipelineError`

Tracker error class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

Viewer

class `parsing.library.viewer.ETAProgressbar`

Bases: `parsing.library.viewer.Viewer`

receive(*tracker*, *broadcast_type*)

Incremental updates of tracking info.

Parameters

- **tracker** (*Tracker*) – Tracker instance.
- **broadcast_type** (*str*) – Broadcast type emitted by tracker.

report(*tracker*)
Do nothing.

class `parsing.library.viewer.Hoarder`
Bases: `parsing.library.viewer.Viewer`

Accumulate a log of some properties of the tracker.

receive(*tracker*, *broadcast_type*)
Receive an update from a tracker.

Ignore all broadcasts that are not TIME.

Parameters

- **tracker** (`parsing.library.tracker.Tracker`) – Tracker receiving update from.
- **broadcast_type** (*str*) – Broadcast message from tracker.

report(*tracker*)
Do nothing.

property `schools`
Get schools attribute (i.e. `self.schools`).

Returns Value of schools storage value.

Return type `dict`

class `parsing.library.viewer.StatProgressBar`(*stat_format*=", *statistics*=None)
Bases: `parsing.library.viewer.Viewer`

Command line progress bar viewer for data pipeline.

SWITCH_SIZE = 100

receive(*tracker*, *broadcast_type*)
Incremental update to progress bar.

report(*tracker*)
Do nothing.

class `parsing.library.viewer.StatView`
Bases: `parsing.library.viewer.Viewer`

Keeps view of statistics of objects processed pipeline.

KINDS

The kinds of objects that can be tracked. TODO - move this to a shared space w/Validator

Type `tuple`

LABELS

The status labels of objects that can be tracked.

Type `tuple`

stats

The view itself of the stats.

Type `dict`

KINDS = ('course', 'section', 'meeting', 'evaluation', 'offering', 'eval')

LABELS = ('valid', 'created', 'new', 'updated', 'total')

receive(*tracker*, *broadcast_type*)

Receive an update from a tracker.

Ignore all broadcasts that are not STATUS.

Parameters

- **tracker** (`parsing.library.tracker.Tracker`) – Tracker receiving update from.
- **broadcast_type** (*str*) – Broadcast message from tracker.

report(*tracker=**None*)

Dump stats.

class `parsing.library.viewer.TimeDistributionView`

Bases: `parsing.library.viewer.Viewer`

Viewer to analyze time distribution.

Calculates granularity and holds report and 12, 24hr distribution.

distribution

Contains counts of 12 and 24hr sightings.

Type `dict`

granularity

Time granularity of viewed times.

Type `int`

receive(*tracker*, *broadcast_type*)

Receive an update from a tracker.

Ignore all broadcasts that are not TIME.

Parameters

- **tracker** (`parsing.library.tracker.Tracker`) – Tracker receiving update from.
- **broadcast_type** (*str*) – Broadcast message from tracker.

report(*tracker*)

Do nothing.

class `parsing.library.viewer.Timer`(*format*='%(elapsed)s', ***kwargs*)

Bases: `progressbar.widgets.FormatLabel`, `progressbar.widgets.TimeSensitiveWidgetBase`

Custom timer created to take away ‘Elapsed Time’ string.

INTERVAL = `datetime.timedelta(microseconds=100000)`

check_size(*progress*)

```
mapping = {'elapsed': ('total_seconds_elapsed', <function format_time>),
'finished': ('end_time', None), 'last_update': ('last_update_time', None), 'max':
('max_value', None), 'seconds': ('seconds_elapsed', None), 'start': ('start_time',
None), 'value': ('value', None)}
```

required_values = []

class `parsing.library.viewer.Viewer`

Bases: `object`

A view that is updated via a tracker object broadcast or report.

abstract receive(*tracker*, *broadcast_type*)

Incremental updates of tracking info.

Parameters

- **tracker** (*Tracker*) – Tracker instance.
- **broadcast_type** (*str*) – Broadcast type emitted by tracker.

abstract report(*tracker*)

Report all tracked info.

Parameters **tracker** (*Tracker*) – Tracker instance.

exception `parsing.library.viewer.ViewerError`(*data*, **args*)

Bases: `parsing.library.exceptions.PipelineError`

Viewer error class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

Digester

class `parsing.library.digester.Absorb`(*school*, *meta*)

Bases: `parsing.library.digester.DigestionStrategy`

Load valid data into Django db.

meta

Meta-information to use for DataUpdate object

Type `dict`

school

Type `str`

classmethod `digest_section`(*parmams*, *clean=True*)

static `remove_offerings`(*section_obj*)

Remove all offerings associated with a section.

Parameters **section_obj** (*Section*) – Description

static `remove_section`(*section_code*, *course_obj*)

Remove section specified from database.

Parameters

- **section** (*dict*) – Description
- **course_obj** (*Course*) – Section part of this course.

wrap_up()

Update time updated for school at wrap_up of parse.

class `parsing.library.digester.Burp`(*school*, *meta*, *output=None*)

Bases: `parsing.library.digester.DigestionStrategy`

Load valid data into Django db and output diff between input and db data.

absorb

Digestion strategy.

Type *Vommit*

vommit

Digestion strategy.

Type *Absorb*

wrap_up()

Do whatever needs to be done to wrap_up digestion session.

class `parsing.library.digester.DigestionAdapter`(*school, cached, short_course_weeks_limit*)

Bases: `object`

Converts JSON defitions to model compliant dictionay.

cache

Caches Django objects to avoid redundant queries.

Type `dict`

school

School code.

Type `str`

adapt_course(*course*)

Adapt course for digestion.

Parameters `course` (*dict*) – course info

Returns Adapted course for django object.

Return type `dict`

Raises *DigestionError* – course is None

adapt_evaluation(*evaluation*)

Adapt evaluation to model dictionary.

Parameters `evaluation` (*dict*) – validated evaluation.

Returns Description

Return type `dict`

adapt_meeting(*meeting, section_model=None*)

Adapt meeting to Django model.

Parameters

- `meeting` (*TYPE*) – Description
- `section_model` (*None, optional*) – Description

Yields `dict`

Raises *DigestionError* – meeting is None.

adapt_section(*section, course_model=None*)

Adapt section to Django model.

Parameters

- `section` (*TYPE*) – Description

- **course_model** (*None, optional*) – Description

Returns formatted section dictionary

Return type `dict`

Raises *DigestionError* – Description

exception `parsing.library.digester.DigestionError`(*data, *args*)

Bases: *parsing.library.exceptions.PipelineError*

Digester error class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class `parsing.library.digester.DigestionStrategy`

Bases: `object`

abstract wrap_up()

Do whatever needs to be done to wrap_up digestion session.

class `parsing.library.digester.Digester`(*school, meta, tracker=<parsing.library.tracker.NullTracker object>*)

Bases: `object`

Digester in data pipeline.

adapter

Adapts

Type *DigestionAdapter*

cache

Caches recently used Django objects to be used as foreign keys.

Type `dict`

data

The data to be digested.

Type `TYPE`

meta

meta data associated with input data.

Type `dict`

MODELS

mapping from object type to Django model class.

Type `dict`

school

School to digest.

Type `str`

strategy

Load and/or diff db depending on strategy

Type *DigestionStrategy*

tracker

Description

Type *parsing.library.tracker.Tracker*

```
MODELS = {'course': <class 'timetable.models.Course'>, 'evaluation': <class  
'timetable.models.Evaluation'>, 'offering': <class 'timetable.models.Offering'>,  
'section': <class 'timetable.models.Section'>, 'semester': <class  
'timetable.models.Semester'>}
```

digest(*data*, *diff=True*, *load=True*, *output=None*)
Digest data.

digest_course(*course*)
Create course in database from info in json model.

Returns django course model object

digest_eval(*evaluation*)
Digest evaluation.

Parameters *evaluation* (*dict*) –

digest_meeting(*meeting*, *section_model=None*)
Create offering in database from info in model map.

Parameters *section_model* – JSON course model object

Return: Offerings as generator

digest_section(*section*, *course_model=None*)
Create section in database from info in model map.

Parameters *course_model* – django course model object

Keyword Arguments *clean* (*boolean*) – removes course offerings associated with section if set

Returns django section model object

wrap_up()

class *parsing.library.digester.Vommit*(*output*)
Bases: *parsing.library.digester.DigestionStrategy*

Output diff between input and db data.

diff(*kind*, *inmodel*, *dbmodel*, *hide_defaults=True*)
Create a diff between input and existing model.

Parameters

- **kind** (*str*) – kind of object to diff.
- **inmodel** (*model*) – Description
- **dbmodel** (*model*) – Description
- **hide_defaults** (*bool*, *optional*) – hide values that are defaulted into db

Returns Diff

Return type *dict*

static **get_model_defaults**()

remove_defaulted_keys(*kind*, *dct*)

wrap_up()

Do whatever needs to be done to wrap_up digestion session.

Exceptions

exception `parsing.library.exceptions.ParseError(data, *args)`

Bases: `parsing.library.exceptions.PipelineError`

Parser error class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `parsing.library.exceptions.ParseJump(data, *args)`

Bases: `parsing.library.exceptions.PipelineWarning`

Parser exception used for control flow.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `parsing.library.exceptions.ParseWarning(data, *args)`

Bases: `parsing.library.exceptions.PipelineWarning`

Parser warning class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `parsing.library.exceptions.PipelineError(data, *args)`

Bases: `parsing.library.exceptions.PipelineException`

Data-pipeline error class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `parsing.library.exceptions.PipelineException(data, *args)`

Bases: `Exception`

Data-pipeline exception class.

Should never be constructed directly. Use:

- `PipelineError`
- `PipelineWarning`

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `parsing.library.exceptions.PipelineWarning(data, *args)`

Bases: `parsing.library.exceptions.PipelineException, UserWarning`

Data-pipeline warning class.

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

Extractor

`class parsing.library.extractor.Extraction(key, container, patterns)`

Bases: `tuple`

container

Alias for field number 1

`count(value, /)`

Return number of occurrences of value.

`index(value, start=0, stop=9223372036854775807, /)`

Return first index of value.

Raises `ValueError` if the value is not present.

key

Alias for field number 0

patterns

Alias for field number 2

`parsing.library.extractor.extract_info_from_text(text, inject=None, extractions=None, use_lowercase=True, splice_text=True)`

Attempt to extract info from text and put it into course object.

NOTE: Currently unstable and unused as it introduces too many bugs. Might reconsider for later use.

Parameters

- **text** (*str*) – text to attempt to extract information from
- **extractions** (*None, optional*) – Description
- **inject** (*None, optional*) – Description
- **use_lowercase** (*bool, optional*) – Description

Returns the text trimmed of extracted information

Return type `str`

Utils

`class parsing.library.utils.DotDict(dct)`

Bases: `dict`

Dot notation access for dictionary.

Supports set, get, and delete.

Examples

```
>>> d = DotDict({'a': 1, 'b': 2, 'c': {'ca': 31}})
>>> d.a, d.b
(1, 2)
>>> d['a']
1
>>> d['a'] = 3
>>> d.a, d['b']
(3, 2)
>>> d.c.ca, d.c['ca']
(31, 31)
```

as_dict()

Return pure dictionary representation of self.

clear() → None. Remove all items from D.

copy() → a shallow copy of D

fromkeys(*value=None, /*)

Create a new dictionary with keys from iterable and values set to value.

get(*key, default=None, /*)

Return the value for key if key is in the dictionary, else default.

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k[, d]*) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise `KeyError` is raised

popitem()

Remove and return a (key, value) pair as a 2-tuple.

Pairs are returned in LIFO (last-in, first-out) order. Raises `KeyError` if the dict is empty.

setdefault(*key, default=None, /*)

Insert key with a value of default if key is not in the dictionary.

Return the value for key if key is in the dictionary, else default.

update(*[E], **F*) → None. Update D from dict/iterable E and F.

If E is present and has a `.keys()` method, then does: for k in E: D[k] = E[k] If E is present and lacks a `.keys()` method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

class `parsing.library.utils.SimpleNamespace`(***kwargs*)

Bases: `object`

`parsing.library.utils.clean`(*dirt*)

Recursively clean json-like object.

list::

- remove *None* elements
- *None* on empty list

dict::

- filter out None valued key, value pairs
- *None* on empty dict

str::

- convert unicode whitespace to ascii
- strip extra whitespace
- None on empty string

Parameters *dict* – the object to clean

Returns Cleaned *dict*, cleaned *list*, cleaned *string*, or pass-through.

`parsing.library.utils.dict_filter_by_dict(a, b)`

Filter dictionary a by b.

dict or set Items or keys must be string or regex. Filters at arbitrary depth with regex matching.

Parameters

- **a** (*dict*) – Dictionary to filter.
- **b** (*dict*) – Dictionary to filter by.

Returns Filtered dictionary

Return type *dict*

`parsing.library.utils.dict_filter_by_list(a, b)`

`parsing.library.utils.dir_to_dict(path)`

Recursively create nested dictionary representing directory contents.

Parameters *path* (*str*) – The path of the directory.

Returns Dictionary representation of the directory.

Return type *dict*

`parsing.library.utils.is_short_course(date_start, date_end, short_course_weeks_limit)`

Checks whether a course's duration is longer than a short term course week limit or not. Limit is defined in the config file for the corresponding school.

Parameters

- **{str}** -- Any reasonable date value for start date (*date_start*) –
- **{str}** -- Any reasonable date value for end date (*date_end*) –
- **{int}** -- Number of weeks a course can be (*short_course_weeks_limit*) –
- as "short term". (*defined*) –

Raises

- **ValidationError** – Invalid date input
- **ValidationError** – Invalid date input

Returns bool – Defines whether the course is short term or not.

`parsing.library.utils.iterrify(x)`

Create iterable object if not already.

Will wrap *str* types in extra iterable eventhough *str* is iterable.

Examples

```
>>> for i in iterrify(1):
...     print(i)
1
>>> for i in iterrify([1]):
...     print(i)
1
>>> for i in iterrify('hello'):
...     print(i)
'hello'
```

`parsing.library.utils.make_list(x=None)`

Wrap in list if not list already.

If input is None, will return empty list.

Parameters *x* – Input.

Returns Input wrapped in list.

Return type *list*

`parsing.library.utils.pretty_json(obj)`

Prettify object as JSON.

Parameters *obj* (*dict*) – Serializable object to JSONify.

Returns Prettified JSON.

Return type *str*

`parsing.library.utils.safe_cast(val, to_type, default=None)`

Attempt to cast to specified type or return default.

Parameters

- **val** – Value to cast.
- **to_type** – Type to cast to.
- **default** (*None*, *optional*) – Description

Returns Description

Return type *to_type*

`parsing.library.utils.short_date(date)`

Convert input to `%m-%d-%y` format. Returns None if input is None.

Parameters *date* (*str*) – date in reasonable format

Returns Date in format `%m-%d-%y` if the input is not None.

Return type *str*

Raises *ParseError* – Unparseable time input.

`parsing.library.utils.time24(time)`

Convert time to 24hr format.

Parameters `time` (*str*) – time in reasonable format

Returns 24hr time in format hh:mm

Return type *str*

Raises `ParseError` – Unparseable time input.

`parsing.library.utils.titlize(name)`

Format name into pretty title.

Will uppercase roman numerals. Will lowercase conjunctions and prepositions.

Examples

```
>>> titlize('BIOLOGY OF CANINES II')
Biology of Canines II
```

`parsing.library.utils.update(d, u)`

Recursive update to dictionary w/o overwriting upper levels.

Examples

```
>>> update({0: {1: 2, 3: 4}}, {1: 2, 0: {5: 6, 3: 7}})
{0: {1: 2}}
```

Parsing Models Documentation

`class parsing.models.DataUpdate(*args, **kwargs)`

Stores the date/time that the school's data was last updated.

Scheduled updates occur when digestion into the database completes.

school

the school code that was updated (e.g. jhu)

Type CharField

semester

the semester for the update

Type ForeignKey to Semester

last_updated

the datetime last updated

Type DateTimeField

reason

the reason it was updated (default Scheduled Update)

Type CharField

update_type

which field was updated

Type CharField

UPDATE_TYPE

Update types allowed.

Type tuple of tuple

COURSES

Update type.

Type str

EVALUATIONS

Update type.

Type str

MISCELLANEOUS

Update type.

Type str

exception DoesNotExist

exception MultipleObjectsReturned

Scheduled Tasks

1.3.13 Frontend Documentation

The Structure

Note: to understand the file structure, it is best to complete the following tutorial: [EggHead Redux](#). We follow the same structure and conventions which are typical of React/Redux applications.

Our React/Redux frontend can be found in `static/js/redux` and has the following structure:

```
static/js/redux
├── __fixtures__
├── __test_utils__
├── __tests__
├── actions
├── constants
├── helpers
├── init.jsx
├── reducers
├── ui
└── util.jsx
```

Let's break down this file structure a bit by exploring what lives in each section.

`__fixtures__`: JSON fixtures used as props to components during tests.

`__test_utils__`: mocks and other utilities helpful for testing.

`__tests__`: unit tests, snapshot tests, all frontend driven tests.

`actions`: all Redux/Thunk actions dispatched by various components. More info on this (more info on this below: *Actions*)

constants: application-wide constant variables

init.jsx: handles application initialization. Handles flows (see [Flows Documentation](#)), the passing of initial data to the frontend, and on page load methods.

reducers: Redux state reducers. (To understand what part of state each reducer handles, see [Reducers](#)).

ui: all components and containers. (For more info see [What Components Live Where](#)).

util.jsx: utility functions useful to the entire application.

Init.jsx

This file is responsible for the initialization of the application. It creates a Redux store from the root reducer, then takes care of all initialization. Only in `init.jsx` do we reference JSON passed from the backend via `timetable.html`.

It is this JSON, called `initData` which we read into state as our initial state for the redux application. However, sometimes there are special *flows* that a user could follow that might change the initial state of the application at page load. For this we use flows which are documented more thoroughly at the following link: [Flows Documentation](#).

Other actions required for page initialization are also dispatched from `init.jsx` including those which load cached timetables from the browser, alerts that show on page load, the loading of user's timetables if logged in, and the triggering of the user agreement modal when appropriate.

Finally, `init.jsx` renders `<Semesterly />` to the DOM. This is the root of the application.

Actions

The actions directory follows this structure:

```
static/js/redux/actions
├── calendar_actions.jsx — exporting the calendar (ical, google)
├── exam_actions.jsx — final exam scheduling/sharing
├── modal_actions.jsx — opening/closing/manipulating all modals
├── school_actions.jsx — getting school info
├── search_actions.jsx — search/adv search
├── timetable_actions.jsx — fetching/loading/manipulating timetables
└── user_actions.jsx — user settings/friends/logged in functionality
```

Reducers

The reducers directory follows this structure:

```
static/js/redux/reducers
├── alerts_reducer.jsx — visibility of alerts
├── calendar_reducer.jsx
├── classmates_reducer.jsx
├── course_info_reducer.jsx
├── course_sections_reducer.jsx
├── custom_slots_reducer.jsx
├── exploration_modal_reducer.jsx
├── final_exams_modal_reducer.jsx
├── friends_reducer.jsx
└── integration_modal_reducer.jsx
```

(continues on next page)

(continued from previous page)

```
├── integrations_reducer.jsx
├── notification_token_reducer.jsx
├── optional_courses_reducer.jsx
├── peer_modal_reducer.jsx
├── preference_modal_reducer.jsx
├── preferences_reducer.jsx
├── root_reducer.jsx
├── save_calendar_modal_reducer.jsx
├── saving_timetable_reducer.jsx
├── school_reducer.jsx
├── search_results_reducer.jsx
├── semester_reducer.jsx
├── signup_modal_reducer.jsx
├── terms_of_service_banner_reducer.jsx
├── terms_of_service_modal_reducer.jsx
├── timetables_reducer.jsx
├── ui_reducer.jsx
├── user_acquisition_modal_reducer.jsx
└── user_info_reducer.jsx
```

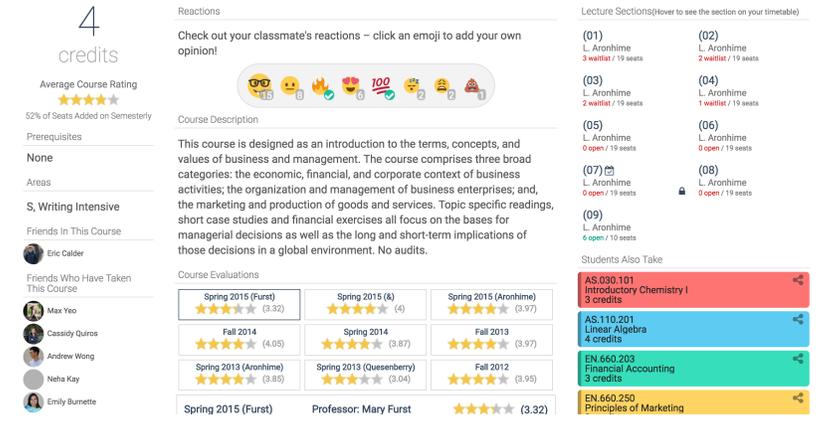
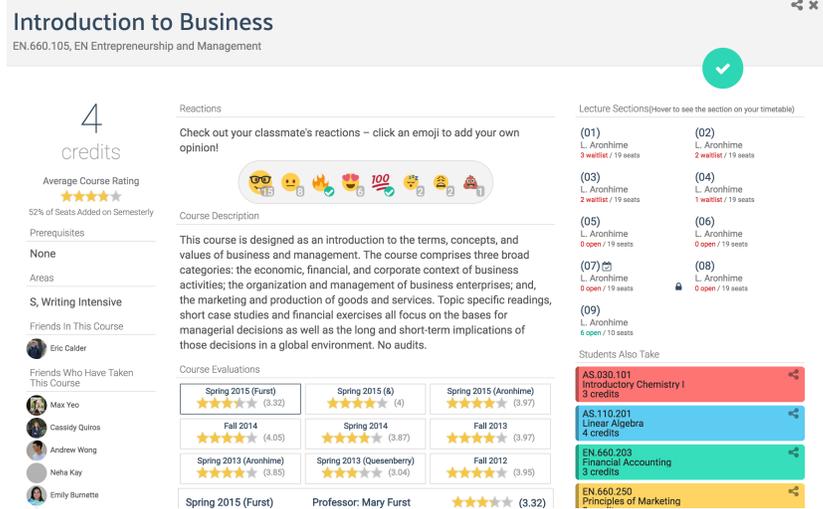
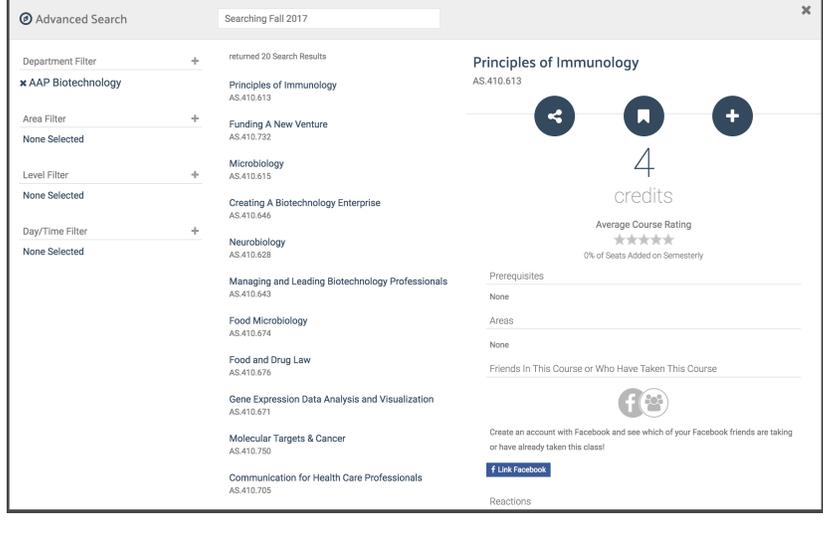
What Components Live Where

All of the components live under the `/ui` directory which follow the following structure:

```
static/js/redux/ui
├── alerts
│   └── ...
├── containers
│   └── ...
├── modals
│   └── ...
└── ...
```

General components live directly under `/ui/` and their containers live under `/ui/containers`. However alerts (those little popups that show up in the top right of the app), live under `/ui/alerts`, and all modals live under `/ui/modals`. Their containers live under their respective sub-directories.

Modals

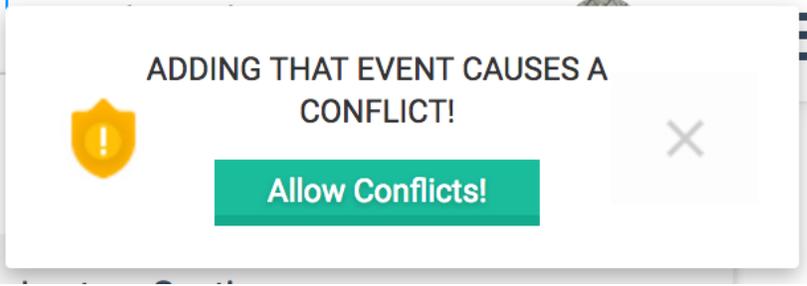
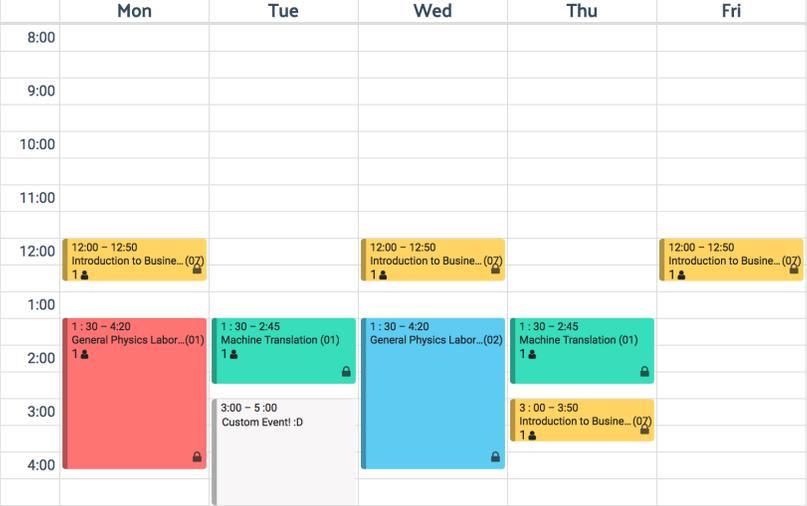
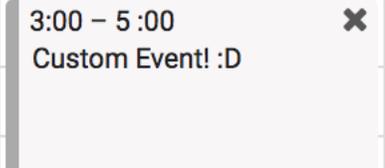
Component File	Screenshot	De- scrip- tion
<p>course_modal_body. jsx</p>		
<p>course_modal. jsx</p>		
<p>exploration_modal. jsx</p>		

1.3. Students know best

Final Exam Schedule

Sun 5/7	Mon 5/8	Tue 5/9	Wed 5/10	Thu 5/11	Fri 5/12	Sat 5/13
				2pm-5pm Human Origins	9am-12pm Data Mining	2pm-5pm Ancient Mythology

General Components

Component File	Screenshot	De- scrip- tion
alert.jsx		
Calendar.tsx		
course_modal_section jsx	<p>Lecture Sections(Hover to see the section on your timetable)</p> <hr/> <p>(01)  P. Koehn 11 waitlist / 30 seats </p>	
CreditTicker.tsx	<p>9.00 credits</p>	
CustomSlot.tsx		
1.3. Students know best CustomSlot.tsx		77

1.3.14 HTML/SCSS Documentation

Note: Although we write SCSS, you'll notice we use the SassLint tool and Sassloader. SASS is an older version of SCSS and SCSS still uses the old SASS compiler. Please don't write SASS, we're a SCSS shop. [You can read about it briefly here.](#)

What's in SCSS, What's not?

Written in SCSS:

1. Web Application

Written in plain CSS:

1. Splash pages
2. Pages for SEO
3. basically everything that is not the web app

File Structure

All of our SCSS is in `static/css/timetable` and is broken down into 5 folders. The `main.scss` ties all the other SCSS files together importing them in the correct order.

Folder	Use
Base	<code>colors.scss</code> and <code>fonts.scss</code>
Vendors	any scss that came from a package that we wanted to customize heavily
Framework	<code>grid.scss</code> and <code>page_layout.scss</code>
Modules	styles for modular parts of our UI
Partials	component specific styles

All of the other CSS files in the `static/css` folder is either used for various purposes outlined above.

Linting and Codestyle

Note: Although we write SCSS, you'll notice we use the SassLint tool and Sassloader. SASS is an older version of SCSS and SCSS still uses the old SASS compiler. Please don't write SASS, we're a SCSS shop. [You can read about it briefly here.](#)

We use SASSLint with Airbnb's `.scss-lint.yml` file converted into `.sass-lint.yml`. Some things to take note of are

1. All colors must be declared as variables in `colors.scss`. Try your best to use the existing colors in that file
2. Double quotes
3. Keep nesting below 3 levels, use BEM
4. Use shortened property values when possible, i.e. `margin: 0 3px` instead of `margin: 0 3px 0 3px`
5. If a property is `0` don't specify units

Refer to our `.sass-lint.yml` for more details and if you're using IntelliJ or some IDE, use the sass-lint module to highlight code-style errors/warnings as you code.

1.3.15 Design/Branding Guidelines

Fonts & Colors

LOGOMARK



TYPOGRAPHY

Palanquin Medium
A a B b C c D d E e F f G g 1 2 3 4 5 6 7 8 9 0

Palanquin Bold
A a B b C c D d E e F f G g 1 2 3 4 5 6 7 8 9 0

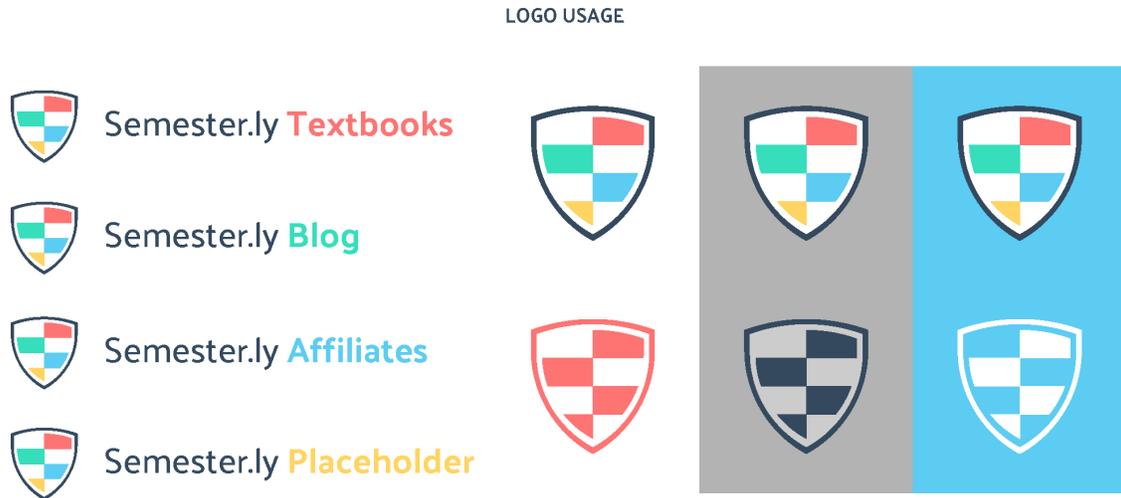
Roboto
A a B b C c D d E e F f G g 1 2 3 4 5 6 7 8 9 0

COLOR PALETTE



#34495E #FD7473 #36DEBB #5CCCF2 #FFD462

Logo Usage



CLEARANCE



Logos

You can download logos and favicon files [here](#)

1.3.16 Editing This Documentation

Building the Docs

From the *docs* directory, execute the following command to rebuild all edited pages:

```
make html
```

To rebuild all pages, you may want to do a clean build:

```
make clean && make html
```

Viewing the Docs Locally

From the docs directory, open the index file from the build directory with the command:

```
open _build/html/index.html
```

Editing the Docs

All Django modules are documented via [Sphinx AutoDoc](#). To edit this documentation, update the docstrings on the relevant functions/classes.

To update the handwritten docs, edit the relevant `.rst` files which are included by filename from `index.rst`.

Note: Be sure no warnings or errors are printed as output during the build process. Travis will build these docs and the build will fail on error.

PYTHON MODULE INDEX

a

agreement.models, 36
authpipe.utils, 42
authpipe.views, 42

C

courses.serializers, 31
courses.utils, 31
courses.views, 30

h

helpers.decorators, 42
helpers.mixins, 42

p

parsing.library.digestor, 61
parsing.library.exceptions, 65
parsing.library.extractor, 66
parsing.library.ingestor, 47
parsing.library.logger, 53
parsing.library.tracker, 56
parsing.library.utils, 66
parsing.library.validator, 49
parsing.library.viewer, 58
parsing.models, 70

S

searches.utils, 35
searches.views, 35
semesterly.test_utils, 36
student.models, 32
student.serializers, 35
student.utils, 34
student.views, 32

t

timetable.models, 23
timetable.serializers, 28
timetable.utils, 28
timetable.views, 28

A

- Absorb (class in *parsing.library.digester*), 61
- absorb (*parsing.library.digester.Burp* attribute), 61
- accept_tos() (in module *student.views*), 34
- adapt_course() (*parsing.library.digester.DigestionAdapter* method), 62
- adapt_evaluation() (*parsing.library.digester.DigestionAdapter* method), 62
- adapt_meeting() (*parsing.library.digester.DigestionAdapter* method), 62
- adapt_section() (*parsing.library.digester.DigestionAdapter* method), 62
- adapter (*parsing.library.digester.Digester* attribute), 63
- add_course() (*semesterly.test_utils.SeleniumTestCase* method), 36
- add_course_from_course_modal() (*semesterly.test_utils.SeleniumTestCase* method), 36
- add_meeting_and_check_conflict() (in module *timetable.utils*), 29
- add_viewer() (*parsing.library.tracker.NullTracker* method), 56
- add_viewer() (*parsing.library.tracker.Tracker* method), 57
- Agreement (class in *agreement.models*), 36
- Agreement.DoesNotExist, 36
- agreement.models module, 36
- Agreement.MultipleObjectsReturned, 36
- all_courses() (in module *courses.views*), 30
- ALL_KEYS (*parsing.library.ingestor.Ingestor* attribute), 47, 48
- allow_conflicts_add() (*semesterly.test_utils.SeleniumTestCase* method), 36
- areas (*timetable.models.Course* attribute), 24
- args (*parsing.library.digester.DigestionError* attribute), 63
- args (*parsing.library.exceptions.ParseError* attribute), 65
- args (*parsing.library.exceptions.ParseJump* attribute), 65
- args (*parsing.library.exceptions.ParseWarning* attribute), 65
- args (*parsing.library.exceptions.PipelineError* attribute), 65
- args (*parsing.library.exceptions.PipelineException* attribute), 65
- args (*parsing.library.exceptions.PipelineWarning* attribute), 65
- args (*parsing.library.ingestor.IngestionError* attribute), 47
- args (*parsing.library.ingestor.IngestionWarning* attribute), 47
- args (*parsing.library.tracker.TrackerError* attribute), 58
- args (*parsing.library.validator.MultipleDefinitionsWarning* attribute), 49
- args (*parsing.library.validator.ValidationError* attribute), 49
- args (*parsing.library.validator.ValidationWarning* attribute), 50
- args (*parsing.library.viewer.ViewerError* attribute), 61
- as_dict() (*parsing.library.utils.DotDict* method), 67
- assert_custom_event_exists() (*semesterly.test_utils.SeleniumTestCase* method), 37
- assert_friend_image_found() (*semesterly.test_utils.SeleniumTestCase* method), 37
- assert_friend_in_modal() (*semesterly.test_utils.SeleniumTestCase* method), 37
- assert_invisibility() (*semesterly.test_utils.SeleniumTestCase* method), 37
- assert_loader_completes() (*semesterly.test_utils.SeleniumTestCase* method), 37
- assert_n_elements_found() (*semesterly.test_utils.SeleniumTestCase* method), 37

- method), 37
- assert_ptt_const_across_refresh() (semesterly.test_utils.SeleniumTestCase method), 37
- assert_ptt_equals() (semesterly.test_utils.SeleniumTestCase method), 37
- assert_slot_presence() (semesterly.test_utils.SeleniumTestCase method), 37
- associate_students() (in module authpipe.utils), 42
- authpipe.utils module, 42
- authpipe.views module, 42
- ## B
- BRACES (parsing.library.logger.JSONStreamWriter attribute), 54, 55
- break_on_error (parsing.library.ingestor.Ingestor attribute), 47
- break_on_warning (parsing.library.ingestor.Ingestor attribute), 47
- broadcast() (parsing.library.tracker.NullTracker method), 56
- broadcast() (parsing.library.tracker.Tracker method), 57
- BROADCAST_TYPES (parsing.library.tracker.NullTracker attribute), 56
- BROADCAST_TYPES (parsing.library.tracker.Tracker attribute), 57
- Burp (class in parsing.library.digester), 61
- ## C
- cache (parsing.library.digester.DigestionAdapter attribute), 62
- cache (parsing.library.digester.Digester attribute), 63
- campus (timetable.models.Course attribute), 23
- can_potentially_conflict() (in module timetable.utils), 29
- change_ptt_name() (semesterly.test_utils.SeleniumTestCase method), 37
- change_term() (semesterly.test_utils.SeleniumTestCase method), 37
- check_size() (parsing.library.viewer.Timer method), 60
- check_student_token() (in module authpipe.utils), 42
- ClassmateView (class in student.views), 32
- clean() (in module parsing.library.utils), 67
- clear() (parsing.library.ingestor.Ingestor method), 48
- clear() (parsing.library.utils.DotDict method), 67
- clear_search_query() (semesterly.test_utils.SeleniumTestCase method), 37
- clear_tutorial() (semesterly.test_utils.SeleniumTestCase method), 37
- click_off() (semesterly.test_utils.SeleniumTestCase method), 38
- close_adv_search() (semesterly.test_utils.SeleniumTestCase method), 38
- close_course_modal() (semesterly.test_utils.SeleniumTestCase method), 38
- code (timetable.models.Course attribute), 23
- colored_json() (in module parsing.library.logger), 56
- compare_timetable() (semesterly.test_utils.SeleniumTestCase method), 38
- complete_user_settings_basics() (semesterly.test_utils.SeleniumTestCase method), 38
- config (parsing.library.validator.Validator attribute), 50
- container (parsing.library.extractor.Extraction attribute), 66
- converter() (parsing.library.logger.JSONColoredFormatter method), 53
- converter() (parsing.library.logger.JSONFormatter method), 54
- copy() (parsing.library.ingestor.Ingestor method), 48
- copy() (parsing.library.utils.DotDict method), 67
- corequisites (timetable.models.Course attribute), 24
- cores (timetable.models.Course attribute), 24
- count() (parsing.library.extractor.Extraction method), 66
- Course (class in timetable.models), 23
- course (timetable.models.Evaluation attribute), 25
- course (timetable.models.Section attribute), 26
- course (timetable.utils.Slot attribute), 28
- Course.DoesNotExist, 24
- Course.MultipleObjectsReturned, 24
- course_code (timetable.models.Evaluation attribute), 25
- course_code_regex (parsing.library.validator.Validator attribute), 50
- course_name_contains_token() (in module searches.utils), 35
- course_page() (in module courses.views), 30
- course_section_id (timetable.models.Section attribute), 27
- CourseDetail (class in courses.views), 30
- CourseIntegration (class in timetable.models), 25
- CourseIntegration.DoesNotExist, 25
- CourseIntegration.MultipleObjectsReturned, 25
- CourseModal (class in courses.views), 30
- COURSES (parsing.models.DataUpdate attribute), 71
- courses (timetable.utils.Timetable attribute), 29
- courses.serializers

- module, 31
 - courses.utils
 - module, 31
 - courses.views
 - module, 30
 - courses_to_slots() (in module *timetable.utils*), 29
 - CourseSearchList (class in *searches.views*), 35
 - CourseSerializer (class in *courses.serializers*), 31
 - create_custom_event()
 - (*semesterly.test_utils.SeleniumTestCase* method), 38
 - create_friend() (*semesterly.test_utils.SeleniumTestCase* method), 38
 - create_personal_timetable_obj()
 - (*semesterly.test_utils.SeleniumTestCase* method), 38
 - create_ptt() (*semesterly.test_utils.SeleniumTestCase* method), 38
 - create_student() (in module *authpipe.utils*), 42
 - CsrfExemptSessionAuthentication (class in *helpers.mixins*), 42
- ## D
- data (*parsing.library.digester.Digester* attribute), 63
 - DataUpdate (class in *parsing.models*), 70
 - DataUpdate.DoesNotExist, 71
 - DataUpdate.MultipleObjectsReturned, 71
 - day (*timetable.models.Offering* attribute), 26
 - default_msec_format
 - (*parsing.library.logger.JSONColoredFormatter* attribute), 53
 - default_msec_format
 - (*parsing.library.logger.JSONFormatter* attribute), 54
 - default_time_format
 - (*parsing.library.logger.JSONColoredFormatter* attribute), 53
 - default_time_format
 - (*parsing.library.logger.JSONFormatter* attribute), 54
 - delete() (*student.views.UserTimetableView* method), 33
 - delete() (*student.views.UserView* method), 34
 - department (*parsing.library.tracker.NullTracker* property), 56
 - department (*timetable.models.Course* attribute), 24
 - description (*timetable.models.Course* attribute), 23
 - description() (*semesterly.test_utils.SeleniumTestCase* method), 38
 - dict_filter_by_dict() (in module *parsing.library.utils*), 68
 - dict_filter_by_list() (in module *parsing.library.utils*), 68
 - diff() (*parsing.library.digester.Vommit* method), 64
 - digest() (*parsing.library.digester.Digester* method), 64
 - digest_course() (*parsing.library.digester.Digester* method), 64
 - digest_eval() (*parsing.library.digester.Digester* method), 64
 - digest_meeting() (*parsing.library.digester.Digester* method), 64
 - digest_section() (*parsing.library.digester.Absorb* class method), 61
 - digest_section() (*parsing.library.digester.Digester* method), 64
 - DigestionAdapter (class in *parsing.library.digester*), 62
 - DigestionError, 63
 - DigestionStrategy (class in *parsing.library.digester*), 63
 - Digester (class in *parsing.library.digester*), 63
 - dir_to_dict() (in module *parsing.library.utils*), 68
 - DisplayTimetable (class in *timetable.utils*), 28
 - DisplayTimetableSerializer (class in *timetable.serializers*), 28
 - distribution (*parsing.library.viewer.TimeDistributionView* attribute), 60
 - DotDict (class in *parsing.library.utils*), 66
 - driver (*semesterly.test_utils.SeleniumTestCase* attribute), 36
- ## E
- edit_custom_event()
 - (*semesterly.test_utils.SeleniumTestCase* method), 38
 - end() (*parsing.library.ingestor.Ingestor* method), 48
 - end() (*parsing.library.tracker.NullTracker* method), 57
 - end() (*parsing.library.tracker.Tracker* method), 58
 - enforce_csrf() (*helpers.mixins.CsrfExemptSessionAuthentication* method), 42
 - enrolment (*timetable.models.Section* attribute), 26
 - enter() (*parsing.library.logger.JSONStreamWriter* method), 55
 - enter_search_query()
 - (*semesterly.test_utils.SeleniumTestCase* method), 39
 - ETAProgressbar (class in *parsing.library.viewer*), 58
 - Evaluation (class in *timetable.models*), 25
 - Evaluation.DoesNotExist, 25
 - Evaluation.MultipleObjectsReturned, 25
 - EVALUATIONS (*parsing.models.DataUpdate* attribute), 71
 - EvaluationSerializer (class in *courses.serializers*), 31
 - EventSerializer (class in *timetable.serializers*), 28
 - exclusions (*timetable.models.Course* attribute), 24
 - execute_action_expect_alert()
 - (*semesterly.test_utils.SeleniumTestCase* method), 39

`exit()` (*parsing.library.logger.JSONStreamWriter method*), 56

`exit_compare_timetable()` (*semesterly.test_utils.SeleniumTestCase method*), 39

`extract_info_from_text()` (*in module parsing.library.extractor*), 66

Extraction (*class in parsing.library.extractor*), 66

F

FeatureFlowView (*class in helpers.mixins*), 42

`file` (*parsing.library.logger.JSONStreamWriter attribute*), 55

`file_to_json()` (*parsing.library.validator.Validator static method*), 50

`find()` (*semesterly.test_utils.SeleniumTestCase method*), 39

`find_slots_to_fill()` (*in module timetable.utils*), 29

`first` (*parsing.library.logger.JSONStreamWriter attribute*), 55

`follow_and_validate_url()` (*semesterly.test_utils.SeleniumTestCase method*), 39

`follow_share_link_from_slot()` (*semesterly.test_utils.SeleniumTestCase method*), 39

`force_login()` (*in module semesterly.test_utils*), 41

`format()` (*parsing.library.logger.JSONColoredFormatter method*), 53

`format()` (*parsing.library.logger.JSONFormatter method*), 54

`formatException()` (*parsing.library.logger.JSONColoredFormatter method*), 53

`formatException()` (*parsing.library.logger.JSONFormatter method*), 54

`formatMessage()` (*parsing.library.logger.JSONColoredFormatter method*), 53

`formatMessage()` (*parsing.library.logger.JSONFormatter method*), 54

`formatStack()` (*parsing.library.logger.JSONColoredFormatter method*), 53

`formatStack()` (*parsing.library.logger.JSONFormatter method*), 54

`formatTime()` (*parsing.library.logger.JSONColoredFormatter method*), 53

`formatTime()` (*parsing.library.logger.JSONFormatter method*), 54

`from_model()` (*timetable.utils.DisplayTimetable class method*), 28

`fromkeys()` (*parsing.library.ingestor.Ingestor method*), 48

`fromkeys()` (*parsing.library.utils.DotDict method*), 67

`function_returns_true` (*class in semesterly.test_utils*), 41

G

`geneds` (*timetable.models.Course attribute*), 24

`get()` (*courses.views.CourseDetail method*), 30

`get()` (*courses.views.SchoolList method*), 30

`get()` (*parsing.library.ingestor.Ingestor method*), 48

`get()` (*parsing.library.utils.DotDict method*), 67

`get()` (*searches.views.CourseSearchList method*), 35

`get()` (*student.views.ClassmateView method*), 32

`get()` (*student.views.UserTimetableView method*), 33

`get()` (*student.views.UserView method*), 34

`get_avg_rating()` (*timetable.models.Course method*), 24

`get_classmates_from_course_id()` (*in module student.utils*), 34

`get_classmates_from_tts()` (*in module student.utils*), 34

`get_classmates_in_course()` (*in module courses.views*), 30

`get_current_semesters()` (*in module timetable.utils*), 29

`get_custom_event_fields()` (*semesterly.test_utils.SeleniumTestCase method*), 39

`get_day_to_usage()` (*in module timetable.utils*), 29

`get_elements_as_text()` (*semesterly.test_utils.SeleniumTestCase method*), 39

`get_evals()` (*courses.serializers.CourseSerializer method*), 31

`get_feature_flow()` (*courses.views.CourseModal method*), 30

`get_feature_flow()` (*helpers.mixins.FeatureFlowView method*), 42

`get_feature_flow()` (*timetable.views.TimetableLinkView method*), 28

`get_friend_count_from_course_id()` (*in module student.utils*), 35

`get_hour_from_string_time()` (*in module timetable.utils*), 29

`get_hours_minutes()` (*in module timetable.utils*), 29

`get_minute_from_string_time()` (*in module timetable.utils*), 30

`get_model_defaults()` (*parsing.library.digester.Vommit static method*), 64

`get_popularity_percent()` (*courses.serializers.CourseSerializer method*), 31

- [get_queryset\(\)](#) (*student.views.UserTimetablePreferenceView method*), 33
[get_reactions\(\)](#) (*timetable.models.Course method*), 25
[get_rexged_courses\(\)](#) (*courses.serializers.CourseSerializer method*), 31
[get_section_dict\(\)](#) (*in module courses.serializers*), 31
[get_sections_by_section_type\(\)](#) (*in module courses.utils*), 31
[get_student\(\)](#) (*in module student.utils*), 35
[get_student_dict\(\)](#) (*in module student.serializers*), 35
[get_student_tts\(\)](#) (*in module student.utils*), 35
[get_test_url\(\)](#) (*semesterly.test_utils.SeleniumTestCase method*), 39
[get_time_index\(\)](#) (*in module timetable.utils*), 30
[get_timetable_name\(\)](#) (*semesterly.test_utils.SeleniumTestCase method*), 40
[get_viewer\(\)](#) (*parsing.library.tracker.NullTracker method*), 57
[get_viewer\(\)](#) (*parsing.library.tracker.Tracker method*), 58
[get_xproduct_indicies\(\)](#) (*in module timetable.utils*), 30
[granularity](#) (*parsing.library.viewer.TimeDistributionView attribute*), 60
- ## H
- [has_conflict](#) (*timetable.utils.Timetable attribute*), 29
[has_viewer\(\)](#) (*parsing.library.tracker.NullTracker method*), 57
[has_viewer\(\)](#) (*parsing.library.tracker.Tracker method*), 58
[helpers.decorators](#) module, 42
[helpers.mixins](#) module, 42
[Hoarder](#) (*class in parsing.library.viewer*), 59
- ## I
- [img_dir](#) (*semesterly.test_utils.SeleniumTestCase attribute*), 36
[index\(\)](#) (*parsing.library.extractor.Extraction method*), 66
[info](#) (*timetable.models.Course attribute*), 23
[ingest_course\(\)](#) (*parsing.library.ingestor.Ingestor method*), 48
[ingest_eval\(\)](#) (*parsing.library.ingestor.Ingestor method*), 48
[ingest_meeting\(\)](#) (*parsing.library.ingestor.Ingestor method*), 48
[ingest_section\(\)](#) (*parsing.library.ingestor.Ingestor method*), 49
[IngestionError](#), 47
[IngestionWarning](#), 47
[Ingestor](#) (*class in parsing.library.ingestor*), 47
[init_screenshot_dir\(\)](#) (*semesterly.test_utils.SeleniumTestCase method*), 40
[instructor](#) (*parsing.library.tracker.NullTracker property*), 57
[instructors](#) (*timetable.models.Section attribute*), 27
[Integration](#) (*class in timetable.models*), 25
[Integration.DoesNotExist](#), 25
[Integration.MultipleObjectsReturned](#), 25
[INTERVAL](#) (*parsing.library.viewer.Timer attribute*), 60
[is_locked](#) (*timetable.utils.Slot attribute*), 28
[is_optional](#) (*timetable.utils.Slot attribute*), 28
[is_short_course\(\)](#) (*in module parsing.library.utils*), 68
[items\(\)](#) (*parsing.library.ingestor.Ingestor method*), 49
[items\(\)](#) (*parsing.library.utils.DotDict method*), 67
[iterrify\(\)](#) (*in module parsing.library.utils*), 68
- ## J
- [JSONColoredFormatter](#) (*class in parsing.library.logger*), 53
[JSONFormatter](#) (*class in parsing.library.logger*), 53
[JSONStreamWriter](#) (*class in parsing.library.logger*), 54
- ## K
- [key](#) (*parsing.library.extractor.Extraction attribute*), 66
[keys\(\)](#) (*parsing.library.ingestor.Ingestor method*), 49
[keys\(\)](#) (*parsing.library.utils.DotDict method*), 67
[kind_to_validation_function](#) (*parsing.library.validator.Validator attribute*), 50
[KINDS](#) (*parsing.library.validator.Validator attribute*), 50
[KINDS](#) (*parsing.library.viewer.StatView attribute*), 59
- ## L
- [LABELS](#) (*parsing.library.viewer.StatView attribute*), 59
[last_updated](#) (*parsing.models.DataUpdate attribute*), 70
[level](#) (*parsing.library.logger.JSONStreamWriter attribute*), 55
[level](#) (*timetable.models.Course attribute*), 24
[load_schemas\(\)](#) (*parsing.library.validator.Validator class method*), 50
[location](#) (*timetable.models.Offering attribute*), 26
[lock_course\(\)](#) (*semesterly.test_utils.SeleniumTestCase method*), 40

`logical_export()` (in module `student.views`), 34

`login_via_fb()` (`semesterly.test_utils.SeleniumTestCase` method), 40

`login_via_google()` (`semesterly.test_utils.SeleniumTestCase` method), 40

M

`make_list()` (in module `parsing.library.utils`), 69

`mapping` (`parsing.library.viewer.Timer` attribute), 60

`meeting_section` (`timetable.models.Section` attribute), 26

`meta` (`parsing.library.digester.Absorb` attribute), 61

`meta` (`parsing.library.digester.Digester` attribute), 63

`MISCELLANEOUS` (`parsing.models.DataUpdate` attribute), 71

`mode` (`parsing.library.tracker.NullTracker` property), 57

`MODELS` (`parsing.library.digester.Digester` attribute), 63, 64

module

`agreement.models`, 36

`authpipe.utils`, 42

`authpipe.views`, 42

`courses.serializers`, 31

`courses.utils`, 31

`courses.views`, 30

`helpers.decorators`, 42

`helpers.mixins`, 42

`parsing.library.digester`, 61

`parsing.library.exceptions`, 65

`parsing.library.extractor`, 66

`parsing.library.ingestor`, 47

`parsing.library.logger`, 53

`parsing.library.tracker`, 56

`parsing.library.utils`, 66

`parsing.library.validator`, 49

`parsing.library.viewer`, 58

`parsing.models`, 70

`searches.utils`, 35

`searches.views`, 35

`semesterly.test_utils`, 36

`student.models`, 32

`student.serializers`, 35

`student.utils`, 34

`student.views`, 32

`timetable.models`, 23

`timetable.serializers`, 28

`timetable.utils`, 28

`timetable.views`, 28

`MultipleDefinitionsWarning`, 49

N

`n_elements_to_be_found` (class in `semesterly.test_utils`), 41

`name` (`timetable.models.Course` attribute), 23

`name` (`timetable.models.Semester` attribute), 27

`notes` (`timetable.models.Course` attribute), 23

`NullTracker` (class in `parsing.library.tracker`), 56

`num_credits` (`timetable.models.Course` attribute), 24

O

`Offering` (class in `timetable.models`), 25

`Offering.DoesNotExist`, 26

`Offering.MultipleObjectsReturned`, 26

`offerings` (`timetable.utils.Slot` attribute), 28

`open_and_query_adv_search()`

(`semesterly.test_utils.SeleniumTestCase` method), 40

`open_course_modal_from_search()`

(`semesterly.test_utils.SeleniumTestCase` method), 40

`open_course_modal_from_slot()`

(`semesterly.test_utils.SeleniumTestCase` method), 40

P

`ParseError`, 65

`ParseJump`, 65

`ParseWarning`, 65

`parsing.library.digester` module, 61

`parsing.library.exceptions` module, 65

`parsing.library.extractor` module, 66

`parsing.library.ingestor` module, 47

`parsing.library.logger` module, 53

`parsing.library.tracker` module, 56

`parsing.library.utils` module, 66

`parsing.library.validator` module, 49

`parsing.library.viewer` module, 58

`parsing.models` module, 70

`patch()` (`student.views.UserView` method), 34

`patterns` (`parsing.library.extractor.Extraction` attribute), 66

`PersonalEvent` (class in `student.models`), 32

`PersonalEvent.DoesNotExist`, 32

`PersonalEvent.MultipleObjectsReturned`, 32

`PersonalEventView` (class in `student.views`), 33

`PersonalTimetable` (class in `student.models`), 32

`PersonalTimetable.DoesNotExist`, 32

`PersonalTimetable.MultipleObjectsReturned`, 32

- PersonalTimeTablePreferencesSerializer (class in timetable.serializers), 28
- PipelineError, 65
- PipelineException, 65
- PipelineWarning, 65
- pop() (parsing.library.ingestor.Ingestor method), 49
- pop() (parsing.library.utils.DotDict method), 67
- popitem() (parsing.library.ingestor.Ingestor method), 49
- popitem() (parsing.library.utils.DotDict method), 67
- post() (searches.views.CourseSearchList method), 35
- post() (student.views.ReactionView method), 33
- post() (student.views.UserTimetableView method), 34
- post() (timetable.views.TimetableLinkView method), 28
- post() (timetable.views.TimetableView method), 28
- prerequisites (timetable.models.Course attribute), 24
- pretty_json() (in module parsing.library.utils), 69
- professor (timetable.models.Evaluation attribute), 25
- ptt_to_tuple() (semesterly.test_utils.SeleniumTestCase method), 40
- put() (authpipe.views.RegistrationTokenView method), 42
- ## R
- Reaction (class in student.models), 32
- Reaction.DoesNotExist, 32
- Reaction.MultipleObjectsReturned, 32
- ReactionView (class in student.views), 33
- reason (parsing.models.DataUpdate attribute), 70
- receive() (parsing.library.viewer.ETAProgressbar method), 58
- receive() (parsing.library.viewer.Hoarder method), 59
- receive() (parsing.library.viewer.StatProgressbar method), 59
- receive() (parsing.library.viewer.StatView method), 59
- receive() (parsing.library.viewer.TimeDistributionView method), 60
- receive() (parsing.library.viewer.Viewer method), 60
- RedirectToSignupMixin (class in helpers.mixins), 42
- RegistrationToken (class in student.models), 32
- RegistrationToken.DoesNotExist, 32
- RegistrationToken.MultipleObjectsReturned, 32
- RegistrationTokenView (class in authpipe.views), 42
- related_courses (timetable.models.Course attribute), 24
- relative (parsing.library.validator.Validator attribute), 50
- remove_course() (semesterly.test_utils.SeleniumTestCase method), 40
- remove_course_from_course_modal() (semesterly.test_utils.SeleniumTestCase method), 40
- remove_defaulted_keys() (parsing.library.digester.Vommit method), 64
- remove_offerings() (parsing.library.digester.Absorb static method), 61
- remove_section() (parsing.library.digester.Absorb static method), 61
- remove_viewer() (parsing.library.tracker.NullTracker method), 57
- remove_viewer() (parsing.library.tracker.Tracker method), 58
- report() (parsing.library.tracker.NullTracker method), 57
- report() (parsing.library.tracker.Tracker method), 58
- report() (parsing.library.viewer.ETAProgressbar method), 58
- report() (parsing.library.viewer.Hoarder method), 59
- report() (parsing.library.viewer.StatProgressbar method), 59
- report() (parsing.library.viewer.StatView method), 60
- report() (parsing.library.viewer.TimeDistributionView method), 60
- report() (parsing.library.viewer.Viewer method), 61
- required_values (parsing.library.viewer.Timer attribute), 60
- ## S
- safe_cast() (in module parsing.library.utils), 69
- same_as (timetable.models.Course attribute), 24
- save_user_settings() (semesterly.test_utils.SeleniumTestCase method), 40
- schema_validate() (parsing.library.validator.Validator static method), 51
- school (parsing.library.digester.Absorb attribute), 61
- school (parsing.library.digester.DigestionAdapter attribute), 62
- school (parsing.library.digester.Digester attribute), 63
- school (parsing.library.ingestor.Ingestor attribute), 47
- school (parsing.library.tracker.NullTracker property), 57
- school (parsing.models.DataUpdate attribute), 70
- school (timetable.models.Course attribute), 23
- SchoolList (class in courses.views), 30
- schools (parsing.library.viewer.Hoarder property), 59
- score (timetable.models.Evaluation attribute), 25
- search() (in module searches.utils), 35
- search_course() (semesterly.test_utils.SeleniumTestCase method), 41
- searches.utils module, 35
- searches.views module, 35
- Section (class in timetable.models), 26
- section (timetable.models.Offering attribute), 26
- section (timetable.utils.Slot attribute), 28

`Section.DoesNotExist`, 27
`Section.MultipleObjectsReturned`, 27
`section_type` (*timetable.models.Section* attribute), 27
`sections` (*timetable.utils.Timetable* attribute), 29
`sections_are_filled()` (in module *courses.utils*), 31
`SectionSerializer` (class in *courses.serializers*), 31
`seen` (*parsing.library.validator.Validator* attribute), 50
`select_nth_adv_search_result()`
 (*semesterly.test_utils.SeleniumTestCase*
 method), 41
`SeleniumTestCase` (class in *semesterly.test_utils*), 36
`Semester` (class in *timetable.models*), 27
`semester` (*parsing.models.DataUpdate* attribute), 70
`semester` (*timetable.models.Section* attribute), 27
`Semester.DoesNotExist`, 27
`Semester.MultipleObjectsReturned`, 27
`semesterly.test_utils`
 module, 36
`SemesterSerializer` (class in *courses.serializers*), 31
`serializer_class` (*student.views.UserTimetablePreferenceView*
 attribute), 33
`setdefault()` (*parsing.library.ingestor.Ingestor*
 method), 49
`setdefault()` (*parsing.library.utils.DotDict* method),
67
`setUp()` (*semesterly.test_utils.SeleniumTestCase*
 method), 41
`setUpClass()` (*semesterly.test_utils.SeleniumTestCase*
 class method), 41
`share_timetable()` (*semesterly.test_utils.SeleniumTestCase*
 method), 41
`short_date()` (in module *parsing.library.utils*), 69
`SimpleNamespace` (class in *parsing.library.utils*), 67
`size` (*timetable.models.Section* attribute), 26
`skip_duplicates` (*parsing.library.ingestor.Ingestor* at-
 tribute), 47
`Slot` (class in *timetable.utils*), 28
`slots_to_timetables()` (in module *timetable.utils*),
30
`SlotSerializer` (class in *timetable.serializers*), 28
`start()` (*parsing.library.tracker.NullTracker* method),
57
`start()` (*parsing.library.tracker.Tracker* method), 58
`StatProgressBar` (class in *parsing.library.viewer*), 59
`stats` (*parsing.library.tracker.NullTracker* property), 57
`stats` (*parsing.library.viewer.StatView* attribute), 59
`StatView` (class in *parsing.library.viewer*), 59
`strategy` (*parsing.library.digester.Digester* attribute),
63
`Student` (class in *student.models*), 32
`Student.DoesNotExist`, 32
`student.models`
 module, 32
`Student.MultipleObjectsReturned`, 32
`student.serializers`
 module, 35
`student.utils`
 module, 34
`student.views`
 module, 32
`StudentSerializer` (class in *student.serializers*), 35
`summary` (*timetable.models.Evaluation* attribute), 25
`SWITCH_SIZE` (*parsing.library.viewer.StatProgressBar*
 attribute), 59
`switch_to_ptt()` (*semesterly.test_utils.SeleniumTestCase*
 method), 41

T

`take_alert_action()`
 (*semesterly.test_utils.SeleniumTestCase*
 method), 41
`tearDown()` (*semesterly.test_utils.SeleniumTestCase*
 method), 41
`term` (*parsing.library.tracker.NullTracker* property), 57
`text_to_be_present_in_element_attribute` (class
 in *semesterly.test_utils*), 41
`text_to_be_present_in_nth_element` (class in
 semesterly.test_utils), 41
`time` (*parsing.library.tracker.NullTracker* property), 57
`time24()` (in module *parsing.library.utils*), 69
`time_end` (*timetable.models.Offering* attribute), 26
`time_start` (*timetable.models.Offering* attribute), 26
`TimeDistributionView` (class in *parsing.library.viewer*), 60
`timeout` (*semesterly.test_utils.SeleniumTestCase* at-
 tribute), 36
`Timer` (class in *parsing.library.viewer*), 60
`Timetable` (class in *timetable.utils*), 29
`timetable.models`
 module, 23
`timetable.serializers`
 module, 28
`timetable.utils`
 module, 28
`timetable.views`
 module, 28
`TimetableLinkView` (class in *timetable.views*), 28
`TimetableView` (class in *timetable.views*), 28
`titlize()` (in module *parsing.library.utils*), 70
`Tracker` (class in *parsing.library.tracker*), 57
`tracker` (*parsing.library.digester.Digester* attribute), 63
`tracker` (*parsing.library.ingestor.Ingestor* attribute), 47
`tracker` (*parsing.library.validator.Validator* attribute),
50
`TrackerError`, 58
`type_` (*parsing.library.logger.JSONStreamWriter* at-
 tribute), 55

U

- UNICODE_WHITESPACE (*parsing.library.ingestor.Ingestor* attribute), 48
- unstopped_description (*timetable.models.Course* attribute), 23
- update() (*in module parsing.library.utils*), 70
- update() (*parsing.library.ingestor.Ingestor* method), 49
- update() (*parsing.library.utils.DotDict* method), 67
- update_events() (*student.views.UserTimetableView* method), 34
- update_locked_sections() (*in module timetable.utils*), 30
- UPDATE_TYPE (*parsing.models.DataUpdate* attribute), 71
- update_type (*parsing.models.DataUpdate* attribute), 70
- url_matches_regex (*class in semesterly.test_utils*), 41
- UserTimetablePreferenceView (*class in student.views*), 33
- UserTimetableView (*class in student.views*), 33
- UIView (*class in student.views*), 34
- usesTime() (*parsing.library.logger.JSONColoredFormatter* method), 53
- usesTime() (*parsing.library.logger.JSONFormatter* method), 54
- V**
- validate (*parsing.library.ingestor.Ingestor* attribute), 48
- validate() (*parsing.library.validator.Validator* method), 51
- validate_course() (*parsing.library.validator.Validator* method), 51
- validate_course_modal() (*semesterly.test_utils.SeleniumTestCase* method), 41
- validate_course_modal_body() (*semesterly.test_utils.SeleniumTestCase* method), 41
- validate_directory() (*parsing.library.validator.Validator* method), 51
- validate_eval() (*parsing.library.validator.Validator* method), 51
- validate_final_exam() (*parsing.library.validator.Validator* method), 51
- validate_instructor() (*parsing.library.validator.Validator* method), 51
- validate_location() (*parsing.library.validator.Validator* method), 51
- validate_meeting() (*parsing.library.validator.Validator* method), 52
- validate_section() (*parsing.library.validator.Validator* method), 52
- validate_self_contained() (*parsing.library.validator.Validator* method), 52
- validate_subdomain() (*in module helpers.decorators*), 42
- validate_time_range() (*parsing.library.validator.Validator* method), 52
- validate_timeable() (*semesterly.test_utils.SeleniumTestCase* method), 41
- validate_website() (*parsing.library.validator.Validator* static method), 52
- ValidateSubdomainMixin (*class in helpers.mixins*), 42
- ValidationError, 49
- ValidationWarning, 49
- Validator (*class in parsing.library.validator*), 50
- validator (*parsing.library.ingestor.Ingestor* attribute), 48
- values() (*parsing.library.ingestor.Ingestor* method), 49
- values() (*parsing.library.utils.DotDict* method), 67
- Viewer (*class in parsing.library.viewer*), 60
- ViewerError, 61
- Vommit (*class in parsing.library.digester*), 64
- vommit (*parsing.library.digester.Burp* attribute), 62
- W**
- waitlist (*timetable.models.Section* attribute), 26
- waitlist_size (*timetable.models.Section* attribute), 27
- was_full (*timetable.models.Section* attribute), 27
- with_traceback() (*parsing.library.digester.DigestionError* method), 63
- with_traceback() (*parsing.library.exceptions.ParseError* method), 65
- with_traceback() (*parsing.library.exceptions.ParseJump* method), 65
- with_traceback() (*parsing.library.exceptions.ParseWarning* method), 65
- with_traceback() (*parsing.library.exceptions.PipelineError* method), 65
- with_traceback() (*parsing.library.exceptions.PipelineException* method), 65

`with_traceback()` (*parsing.library.exceptions.PipelineWarning method*), 65

`with_traceback()` (*parsing.library.ingestor.IngestionError method*), 47

`with_traceback()` (*parsing.library.ingestor.IngestionWarning method*), 47

`with_traceback()` (*parsing.library.tracker.TrackerError method*), 58

`with_traceback()` (*parsing.library.validator.MultipleDefinitionsWarning method*), 49

`with_traceback()` (*parsing.library.validator.ValidationError method*), 49

`with_traceback()` (*parsing.library.validator.ValidationWarning method*), 50

`with_traceback()` (*parsing.library.viewer.ViewerError method*), 61

`wrap_up()` (*parsing.library.digester.Absorb method*), 61

`wrap_up()` (*parsing.library.digester.Burp method*), 62

`wrap_up()` (*parsing.library.digester.DigestionStrategy method*), 63

`wrap_up()` (*parsing.library.digester.Digester method*), 64

`wrap_up()` (*parsing.library.digester.Vommit method*), 64

`write()` (*parsing.library.logger.JSONStreamWriter method*), 56

`write_key_value()` (*parsing.library.logger.JSONStreamWriter method*), 56

`write_obj()` (*parsing.library.logger.JSONStreamWriter method*), 56

Y

`year` (*parsing.library.tracker.NullTracker property*), 57

`year` (*timetable.models.Evaluation attribute*), 25

`year` (*timetable.models.Semester attribute*), 27