

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light mint green. They are positioned diagonally, with the blue one partially covering the green one.

Final Project

Tactical Hippos

Team Organization





Project 3 Workload Distribution

- Web Scraper - Nathan
- Model - Ian
- View
 - Login / logout - Axay
 - Edit profiles - Axay
 - (register new users) - Axay
 - Search for specific class information - Ryan
 - Display class information - Ryan
- Controller
 - Search - Ryan
 - Login - Srujan
 - Set ups and CRUD - Daqian



Project 4 Work Distribution

- Model
 - Being able to create child models inside parent - Ian
 - Being able to make an instructor entry for when you scrape the page of courses - Axay, Ryan
 - Fix pull request for recommendations table - Nathan
 - Add days of week to section - Ian
 - Fix Scraper by making it use the new create method - Ian
 - Current applications by user (All grader_sections by user and by whether it is an application or a position) - Ryan
 - Assign / unassign grader to section (change column from application to position) - Ryan
 - Add view for submitting applications - Ryan
 - General template system for listing courses/sections - Ryan
 - Make the login page the default and navigate to a different page based on the type of user logged in - Ryan
 - Add a view for submitting a recommendation - Nathan
 - Add a view for seeing all current sections that you are teaching (instructor only) / grading (grader only) - Srujan
 - Add a view for seeing all sections, all instructors, all graders (admin only) - Srujan
 - Add a view for assigning graders to sections (gets all applications, allows you to change from application to position and vice versa) - Srujan
 - Implement the scraper into a view (admin only)
 - Check type of user for specific requests - Daqian
 - Add search methods for different controllers. - Daqian
 - Switch to Postgresql - Nathan

Major Architecture Decisions





The Model

- User
 - User Type
- Course
- Section
 - Section Type
 - Instructor
 - Term
 - Season
 - Instruction Mode
 - Location
- Grader_Section



The Model - User

- User Type
 - Title (e.g. Grader, Instructor, Admin)
- First Name
- Last Name
- Email

Has the data for one logged-in user. Uses the Devise gem to handle login things like usernames and passwords.



The Model - Course

- **Catalogue Number** ("CSE 1222")
- **Title** ("Introduction to Computer Programming in C++ for Engineers and Scientists")
- **Description** ("Introduction to computer programming and to problem solving techniques using computer programs with applications in engineering and the physical sciences; algorithm development; programming lab experience. Concur: Math 1151, 1154, or 1161. Not open to students with credit for Engr 1281.01 or 1281.02. This course is available for EM credit.")

Stores the information for one course. Does not hold any information about the sections within that course.



The Model - Section

- Course ID (what course it belongs to)
- Instructor
 - Name ("Scott Sharkey")
 - User ID (points to the login for the instructor)
- Section Type
 - Title ("Lab", "Lecture")
- Term
 - Season
 - Title ("Autumn", "Winter")
 - Year (2021)
- Instruction Mode
 - Mode ("Online", "In-Person")
- Location
 - Location ("Online", "Dreese Labs 123")
- Section Number (0010)
- Class Number (5088)
- Start Date (January 1, 2021)
- End Date (June 1, 2021)
- Days of Week ("MWF")
- Start Time (12:45 PM)
- End Time (2:00 PM)



The Model - Grader_Section

- User ID
- Section ID
- isApplication (true or false)

This table documents the relationship between graders and sections. There can be many graders per section, and a grader can grade more than one section, so another table was needed. In addition, because an application to be the grader of a section did not require any additional data (like an essay), a boolean was added to show whether this entry was an application (created by the grader) or a position (approved by an admin).



Separating data into sub-models

Any data that could be repeated was stored in its own model (e.g. the semester that a section of a course is in). This was done for two reasons:

1. This ensures the data is clean. If there is one entry that misspells “Autumn,” for example, the extra entry will be easy to see and correct.
2. This ensures that all matching data points to the same thing, so no data is repeated.

The drawback to this is that there are 7 more tables, all of which only store a small amount of data. For a small project like this, it was unnecessary to break everything up like we did. This was the theoretically ideal solution but it made things confusing to work with.

This was very effective for this project, because, for example, with 353 scraped sections, all of which are unique, only 1 season (which they all shared) was created.



Course Listings

With course listings used in many different areas, we used partial renders

Partial renders are layouts that allow for more modular code

While in the HTML, render as if you're in a controller

Because course listings are used in so many different views, we

Problems





Webscraper

The first problem we encountered was webscraping from the original link.

The information from the classes was loaded in with AngularJs after opening the link so an HTTP request using the gem Mechanize returned the HTML without the classes information.

Solution:

We used the gem watir to open an instance of firefox and pause until the class information loaded in, then it made a request to get the HTML code from dynamically loaded page and killed the firefox instance.

After this we used Nokogiri to collect the data from the HTML as usual.



Switching from SQLite3 to PostgreSQL

To be able to deploy on Heroku as a part of project 4 we have to change databases from project 3.

The first thing that needs to be done is add the gem 'pg' to the gemfile and bundle install

Next, we modified the config/database.yml file in order to setup the new database.

After this, rails db: create and rails db:migrate are supposed to work but are giving us errors at the moment.


Once this step is resolved, we must change the migration files to t.text from t.string for PGSQL.



Creating an instructor when scraping a course

When making a section of a course, the instructor that is in the description needs to be stored somewhere. This needs to also be associated with an instructor user, but there is not a login for every instructor that is being scraped.

Solution: we added an Instructor table that has the instructor name and a key to the User table for the associated Instructor login if that is ever made.



Creating all of the associated sub-data when you make a Section

When creating a Section, you want to also create all of the data in the sub-tables that are associated with that Section. For example, making a Section also involves making the Location for that section.

Solution: in the model file, add the line

```
accepts_nested_attributes_for {table}
```

This allows you to pass in the child table as its own object when you create the parent.

```
User.Create(:user_type => UserType.Create(:title => "Instructor"),  
:first_name => "Scott", :last_name => "Sharkey");
```



Making sure there is no repeat data

When creating an entry in the database (especially for the child tables like User Type or Term), you don't want a duplicate entry with the same data, because the whole point of creating child tables is to make sure that all data that means the same thing is actually referring to the same thing.

Solution: use the method `create_or_find_by` when creating an entry. It will either find the existing entry and use that as the foreign key for the parent or it will make a new entry.

```
User.create_or_find_by(:user_type => UserType.create_or_find_by(:title  
=> "Instructor"), :first_name => "Scott", :last_name => "Sharkey");
```



Searching and filtering models

When we are trying to get the search function done, we need to get the data which shares the same attribute sometimes. Softwares like Elastic Search can be powerful but it requires a huge amount of dependencies for our project.

Solution: Defining different scopes inside the model file like

```
Scope :filter_by_instructor, -> (instructor) { where instructor:  
instructor }
```

Then we are able to call

```
@sections = Section.filter_by_instructor"instructor name"
```

Areas for improvement /
changes





Controller Flexibility

A lot of views would benefit from a dedicated controller request endpoint for GETting a large amount of particular sets of data at once. Things like getting all of the sections associated with a course would be very beneficial.



DRY

Don't Repeat Yourself

Initially, different sections of code were used in single areas

These areas had to be modified after the initial writing to be more adaptable, as they became used more often, especially with course listings

While in the end much of the project was done following DRY, it should have been done initially

Beneficial Tools





Devise

- Authentication solution for Rails
- Functionality for creating accounts, sign in, and logout functions
- Devise helpers manage information in controllers
- Modularity concept; only use what you need
- Composed of 10 modules for adding functionality