CS 493 Final Project

Code due at the time of your final project grading demo
(grading demos will occur during finals week; more details TBA)

In this course, a final programming project will take the place of formal exams to test your understanding of the material. The final project will involve working with a team of 3-4 people to implement a complete RESTful API for an application called Tarpaulin. You can find more details about Tarpaulin below. The API you implement will need to utilize most of the components of a modern API that we talked about in class.

Tarpaulin

The application for which you'll write an API for this project is Tarpaulin, a lightweight course management tool that's an "alternative" to Canvas. In particular, Tarpaulin allows users (instructors and students) to see information about the courses they're teaching/taking. It allows instructors to create assignments for their courses, and it allows students to submit solutions to those assignments.

The Tarpaulin API you implement must support all of the endpoints described in the Tarpaulin OpenAPI specification. You can load this specification into the editor at https://editor.swagger.io to see automatically-generated documentation for all the API endpoints you'll need to implement for the final project. Importantly, you are free to implement these endpoints however you see fit. For example, you may use whatever database you want, and you may design your API architecture to meet your own needs.

The OpenAPI specification linked above will provide most of the details of the API you'll implement, but some more details are included below.

Entities

There are several kinds of entity the Tarpaulin API will need to keep track of:

- **Users** – These represent Tarpaulin application users. Each User can have one of three roles: `admin`, `instructor`, and `student`. Each of these roles represents a different set of permissions to perform certain API actions. The permissions associated with these roles are defined further in the Tarpaulin OpenAPI specification.
- **Courses** – These represent courses being managed in Tarpaulin. Each Course has basic information, such as subject code, number, title, instructor, etc. Each Course also has associated data of other entity types, including a list of enrolled students (i.e. Tarpaulin Users with the `student` role) as well as a set of assignments. More details about how to manage these pieces of data are included both below and in the Tarpaulin OpenAPI specification linked above.

- **Assignments** – These represent a single assignment for a Tarpaulin Course. Each Assignment belongs to a specific Course and has basic information such as a title, due date, etc. It also has a list of individual student submissions.
- **Submissions** – These represent a single student submission for an Assignment in Tarpaulin. Each submission belongs both to its Assignment to the student who submitted it, and it is marked with a submission timestamp. Each submission is also associated with a specific file, which will be uploaded to the Tarpaulin API and stored, so it can be downloaded later. Finally, each submission may be assigned a grade, though grades cannot be assigned when the submission is first created but must be assigned later through an update operation.

Actions

Many of the actions that can be performed with the Tarpaulin API are similar to ones we've seen in the work we've done in class, including fetching entity data and creating, modifying, and deleting entities. These actions should not need much explanation beyond what's included in the Tarpaulin OpenAPI specification. A few specific actions deserve more attention, though:

- **Course roster download** – this action, implemented by the `GET /courses/{id}/roster` endpoint, allows certain authorized users to download a CSV-formatted roster for a specific course. The roster will contain a list of the students currently enrolled in the course, in CSV format, e.g.:

  ```
  "abc123","Leia Organa","organal@oregonstate.edu"
  "def456","Luke Skywalker","skywallu@oregonstate.edu"
  ...
  ```

  Importantly, this file must be generated by the API, based on the list of enrolled students stored in the database.

- **Assignment submission creation** – this action, implemented by the `POST /assignments/{id}/submissions` endpoint, allows authorized `student` Users to upload a file submission for a specific assignment. Importantly, the file uploaded for each Submission must be stored by the API in such a way that it can be later downloaded via URL. Specifically, when storing the submission file, the API should generate the URL with which that file can later be accessed. This URL will be returned along with the rest of the information about the Submission from the `GET /assignments/{id}/submissions` endpoint.

- **User data fetching** – this action, implemented by the `GET /users/{id}` endpoint, allows Users to see their own data. Importantly, only a logged-in User can see their own data. The data returned by this endpoint should also include the list of classes the User is enrolled in (for `student` Users) or teaching (for `instructor` Users).

- **Course information fetching** – this action, implemented by the `GET /courses` and `GET /courses/{id}` endpoints, allows users to see information about all Courses or about a specific Course. Note that the information included by both of these endpoints should not return information about a Course's enrolled students or its Assignments. Instead, that information can be fetched by the `GET /courses/{id}/students` and `GET /courses/{id}/assignments` endpoints, respectively.

Pagination

A few of the Tarpaulin API endpoints must be paginated:

- `GET /courses`
- `GET /assignments/{id}/submissions`

It's up to you to determine the appropriate way to paginate these endpoints, including how the page size is set, etc.

Authorization

Many of the endpoints in the Tarpaulin API require authorization, as described in the Tarpaulin OpenAPI specification. You may implement this using the standard JWT-based authorization scheme we discussed in class.

Rate limiting

Your API should be rate-limited as follows:

- For requests made without a valid authentication token, your API should permit 10 requests/minute. These requests should be rate-limited on a per-IP address basis.
- For requests made *with* a valid authentication token, your API should permit 30 requests per minute. These requests should be rate-limited on a per-user basis.

Docker containerization

All services used by your API (e.g. databases, caches, processing pipelines, etc.) should be run in Docker containers. These containers can be manually created and initialized via the command line.

New tech, 3rd-party libraries, and other tools

You may treat the final project as an opportunity to learn how to use API backend technologies we didn't cover in class. Specifically, if there's a database implementation, third-party tool or library, etc. you want to use for the project, feel free to do so.

GitHub repositories

The code for your final project must be in a GitHub repository set up via GitHub Classroom. You can use this link to form your team and create your final project repository:

https://classroom.github.com/a/LEHQu3Rm

The repository created for your team will be private by default. However, you will have full administrative control over the repository that's created for your project, which means you'll be able to make it public if you wish. I encourage you to make your repo public. These final projects should be nice demonstrations of your web development abilities and will be a good item to have in your CS portfolio. It will be great to have the code in a public GitHub repo so you can share it easily when you want to.

If you've already started a GitHub repo for your project, don't worry. The repository created via the GitHub classroom link above will be completely empty, so you can simply use git remotes to work with both repositories. I can help you set that up if needed.

Working with a team on a shared GitHub repo

When working with a team on a shared GitHub repo, it's a good idea to use a workflow that uses branches and pull requests. This has a few advantages:

- By not working within the same branch, you can better avoid causing conflicts, which can occur when you and another member of your team edit the same parts of the code at the same time.
- It helps you to be more familiar with the entire code base, even the parts that other team members are working on, because you'll see all of the changes to the code as you review pull requests. This can help you develop more rapidly because you won't have to spend as much time understanding code that others have written.
- It helps to ensure high quality code. Code in pull requests is not incorporated into the main code branch until the code request is reviewed and approved. That means everyone has a chance to improve pull request code before it becomes permanent.

One simple but effective branch- and pull-request-based workflow you might consider is the GitHub flow: https://guides.github.com/introduction/flow/.

Grading demonstrations

To get a grade for your project, your team must do a brief (10-15 minute) demonstration to me (Hess) of your project's functionality. To get a grade for your project, your team ***must*** do a demo. These demos will be scheduled for finals week. I'll send more details on scheduling demos for the final project when we get closer to that time.

Note that you should have a set of requests/tests written and ready to go when you arrive at your demo. These tests should fully demonstrate your API's functionality. You are free to use Postman, Insomnia, or any other API testing tool you like to implement these tests. Having these tests ready to go at your demo will comprise part of your grade for this project.

Submission

All code for your final project must be pushed to the main branch of the repo created for your team using the GitHub Classroom link above before your grading demo.

Grading criteria

Your team's grade (out of 100 points) for the final project will be based on successfully implementing a complete API that satisfies these criteria:

- 50 points – Your API successfully implements all of the endpoints described in the Tarpaulin OpenAPI specification.
- 10 points – Your API endpoints correctly require authorization, as described in the Tarpaulin OpenAPI specification.
- 5 points – Your API correctly paginates the appropriate endpoints, as described above and in the Tarpaulin OpenAPI specification.
- 5 points – Your API implements rate limiting, as described above.
- 10 points – Your API's services are run in Docker containers, as described above.
- 10 points – You have a set of tests/requests written and ready to go at your grading demo to fully demonstrate your API's functionality.
- 10 points – Your API has a high-quality design and implementation.

Remember also, if your team does not do a demo for your project, you will receive a zero for it.

Individual grades

Your individual grade for the project will be based on your team's grade and also on evidence of your meaningful participation in your team's work on the project, including from these sources:

- The commit log of your GitHub repository.
- Your presence at and participation in your team's project demo.
- A team evaluation completed by each member of your project team.

In particular, if your GitHub commit log shows that you did not make meaningful contributions to your team's implementation of your app, if you do not participate in your team's demonstration of your app (without explicit prior approval by me), or if your project teammates submit team evaluations in which they agrees that you did not do an appropriate share of the work on your final project, you will receive a lower grade on the project than your teammates. I may use other sources as evidence of your participation, as well.