GradeMe

CS428 Software Engineering II

Group Members:

Alvin Chang, Joe Cibicek, Sam Stephens, SeokHyun Song, Xin Cheng

University of Illinois Urbana-Champaign

4/28/2018

Table of Contents

1. Introduction

Educational institutions such as universities need to be able to keep track of grading for a large number of students. Challenges such as keeping track of so many students, courses, assignments, entering grades in batches and viewing grades for a large number of students can be solved with software.

Our project, Grademe (GME), is a tool in the same family as Compass2g, Gradescope, and Moodle, and is in a more graceful way so that it is easy to use. Students, instructors and TA's can create accounts and manage courses, assignments and submissions.

2. Process

Our team is using Extreme Programming (XP) for the project. XP process helps us respond quickly to the changing developing environment and requirements, as well as maintaining good software quality.

The project is completed within 6 iterations. Each iteration lasts two weeks. We use Slack for our primary coordination and hold multiple meetings between iterations to accomplish work and do pair programming. We meet regularly with our TA, demonstrate what we have accomplished, report our process, and discuss new tasks for next iteration. Typically, we hold 3-4 meetings within each iteration. In each meeting, we discuss new features to implement and change plans according to the feedback from TA. As part of the planning game of XP, we discuss which user stories we want to implement based both off of priority and amount of work required. This is primarily where our pair programming is done as well. Especially in the first iterations, both the frontend and backend team did large amounts of pair programming, especially because the tools that were being used were very new. As a result, knowledge and design were the limited factors, not number of keyboards, so this style of programming was very effective.

Two programming groups are formed to implement frontend and backend in parallel. All code is maintained through gitlab, and uses automated testing for every push. Also, our frontend and backend use different repositories, so user story functionalities are spread out.

Our team tries our best to commit with meaningful commit messages to help other team members understand changes that have been made. To create a user story, a new branch is created, changes and test made, pushed, and then merged by a different user on the team after being reviewed. This can create difficulties with collective code ownership, since one or two developers working on a feature may cause others to not really know what's being created. Also, separate repositories can further this division. We took a few steps to mitigate this. Because the frontend and backend have to coordinate with API calls, already there was a requirement that the frontend knows what the backed is developing. Similarly, the backend has to know what the API call is for. Code review, done through meetings and through pull requests, help. A different student would review and merge another student's code whenever possible. Whenever a feature was implemented, we would demo it to each other at our meetings and explain what we did to make sure we understand the feature. We have at least 5 unit tests for each user story. For the project, advanced testing through Selenium is written to ensure the quality

of the project and avoid careless bugs during coding process. This testing enables test-driven development, a key aspect of XP. All code must pass the test before being merged.

We conduct regular code refactoring as our code base grow. For example, we remove nearly duplicated code by extracting the similarity to shared functions, reform a function to make it more readable to reviewers, etc. Refactoring helps our code review process, provides more flexibility for our future implementation, and improves our code reusability. Refactoring allows cleaner code that self-documents more efficiently.,

During the development process, we are moving developers from each end based on the workload in each iteration. After the fourth iteration, it was clear the backend was progressing at a much faster rate than the frontend, since a new API could be tested and written quickly, however, the workload to create a page was still high. So, we moved another developer from the backend to the frontend to compensate, and were able to implement many more user stories than expected.

3. Requirements & Specifications

3.1. Use cases

The GME system supports following primary use cases. There are three primary actors in GME: Studen, TA, and Instructor.

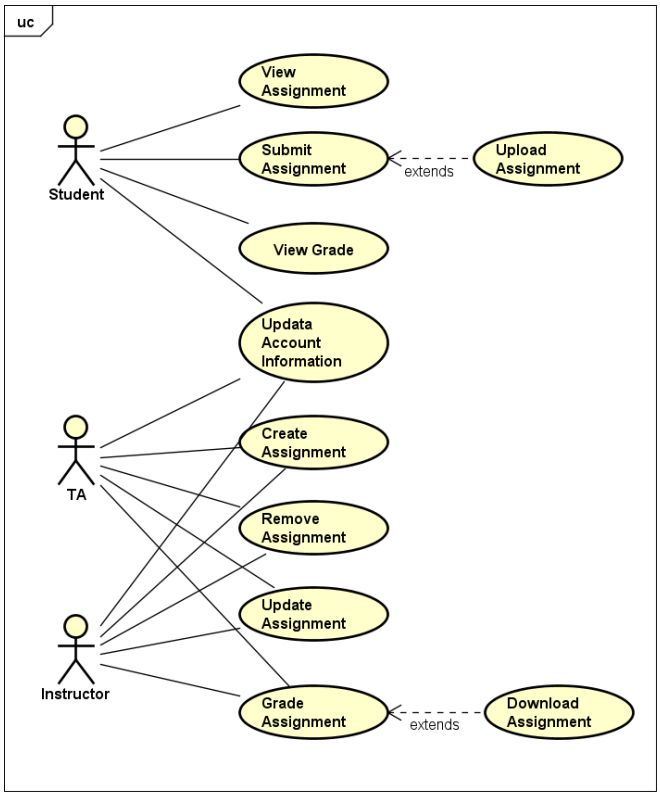

Figure 1 Use case diagram of GME

3.2. Actor-goal list:

Table 1 Actor goal list of GME

Actor	Task-level goal	Priority
All	Login	1
All	Logout	4
All	Reset PW	4
All	Create Account	1
All	Modify Account Setting	4
All	Manage Course	2
Instructor	Create Course	1
Instructor	Manage Course	3
Instructor	Modify Course Page	2
Instructor	CRUD Assignment	2
Instructor	Add/Remove users	2
TA/Instructor	Enter Grade	1
TA/Instructor	Upload Grades	2
TA/Instructor	Grade Assignment	1
Student	View Assignment	1
Student	Submit Assignment	2
Student	View Assignment Description	2
Student	View Course	1
Student	Register Courses (code)	1
Student	View Grades	2

3.3. Brief Use case forms

3.3.1. Student

Submit Assignment

Use case brief: A student uploads a submission if needed and fills out any other information requested by the assignment. They can then save or submit the assignment for grading.

Casual use case: A student viewing the assignment page wants to submit their assignment. An assignment may have questions that require a response to be typed into a textbox, such as short answer or essay questions. Additionally, there may be multiple choice questions. If part of the assignment, the student may also upload one or more files to be used for grading. Files that are too large or invalid type are rejected. After completing the assignment, the student may save the submission for later editing or submit the solution for grading. In either case, the website saves the submission.

View Grades

Use case brief: Student opens up the grading section which displays all the available grades. Then user navigates to a particular grade he is interested in.

Casual use case: Student is interested in viewing their grades. The grading section displays grades for all courses, some assignments are already graded while some of them can have a pending status on them. Additionally, student can use various filters to display the grades, either filter by particular course, date of last grading or by the percentage of a grade.

3.3.2. TA

Enter Grade

Use case brief: A TA/Instructor inputs grade by manually putting score or rubric set up by the instructor. They then save the grade entered.

Casual Use Case: A TA/Instructor selects a specific student. It then goes to a page where the TA/Instructor can put in the points that the student received or if the Instructor set up a rubric, the TA/Instructor can select the grade the student received in each category. The points are then saved.

3.3.3. Instructor

Create Course

Use case brief: Create a course which is open for registration.

Causal use case: The instructor creates a course and provides course information such as: Title, Description, and Prerequisites. GradeMe checks if there exists conflict. If there are no conflict, GradeMe responds with confirmation, saves the course to the system, and then makes it available for students to register. If conflicts occur, GradeMe notifies the instructor and rejects the request.

Modify a Course Page

Use case brief: Modify a course Page which has registered.

Causal use case: The instructor modifies a course page such as: Description, and Prerequisites, Homework date, MP deadline, exam etc. If the update is not logical, GradeMe will check and notifies the instructor and rejects the request; If the update is making sense, GradeMe will tell the instructor and accept the request.

4. Architecture & Design

4.1. UML diagrams

4.1.1. Class diagram

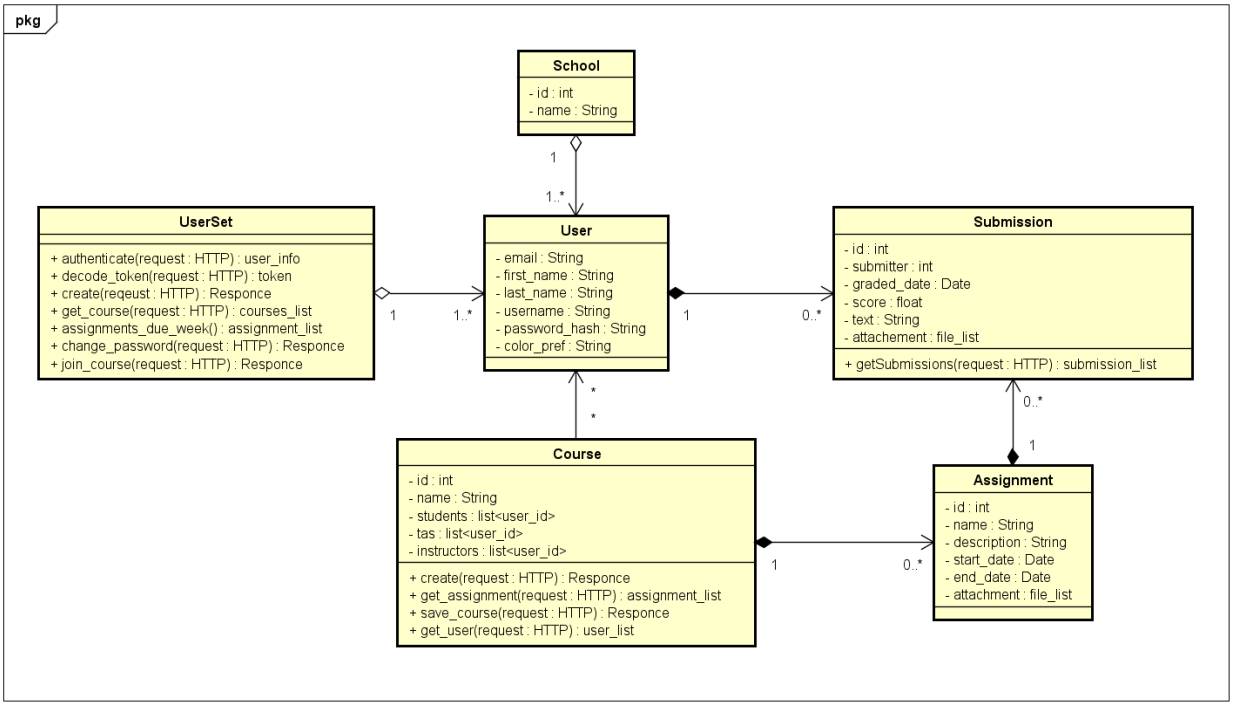

Figure 2 Class diagram of GME

4.1.2. Sequence diagram

Get course

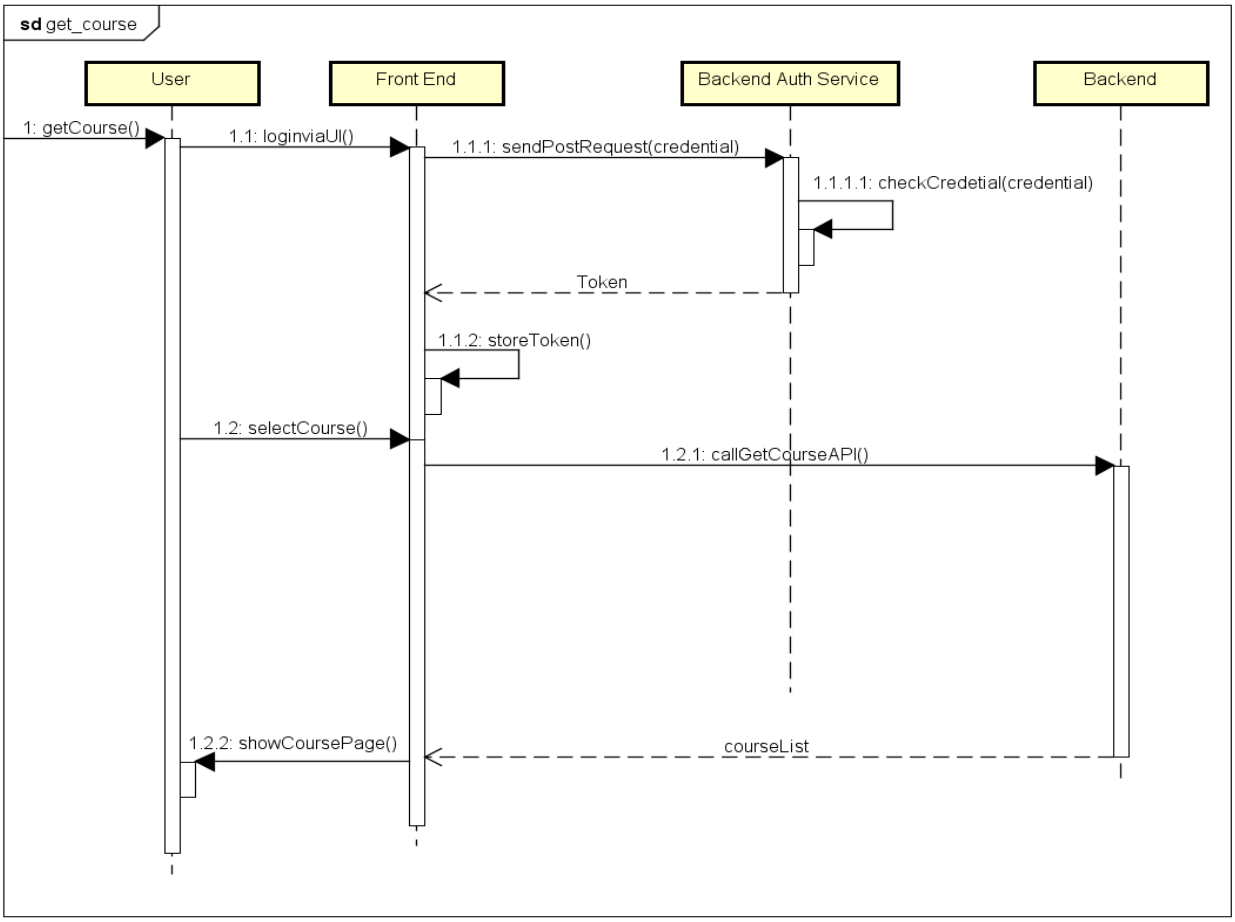

Figure 3.1 Sequence diagram of get course use case

Get Submission

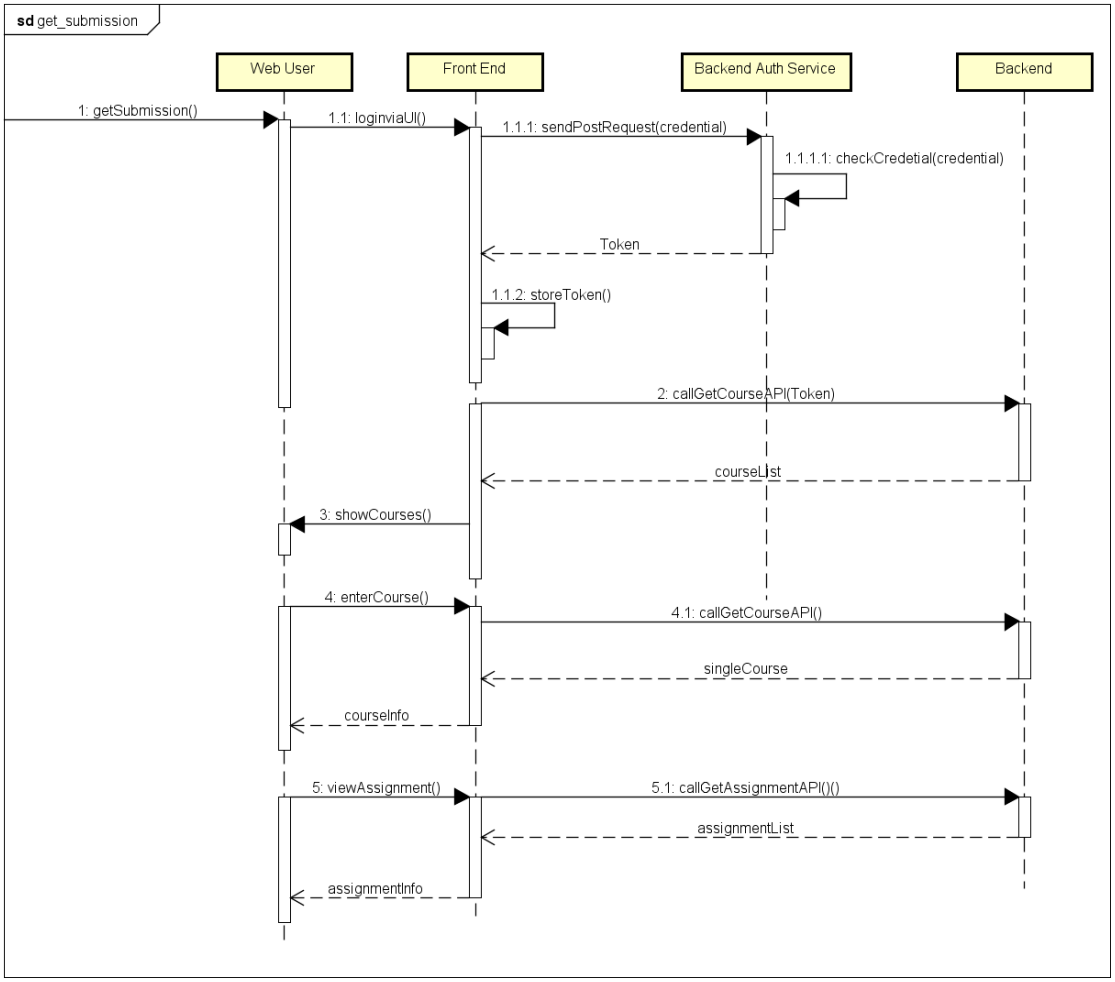

Figure 3.2 Sequence diagram of get submission use case

4.1.3. ER diagram

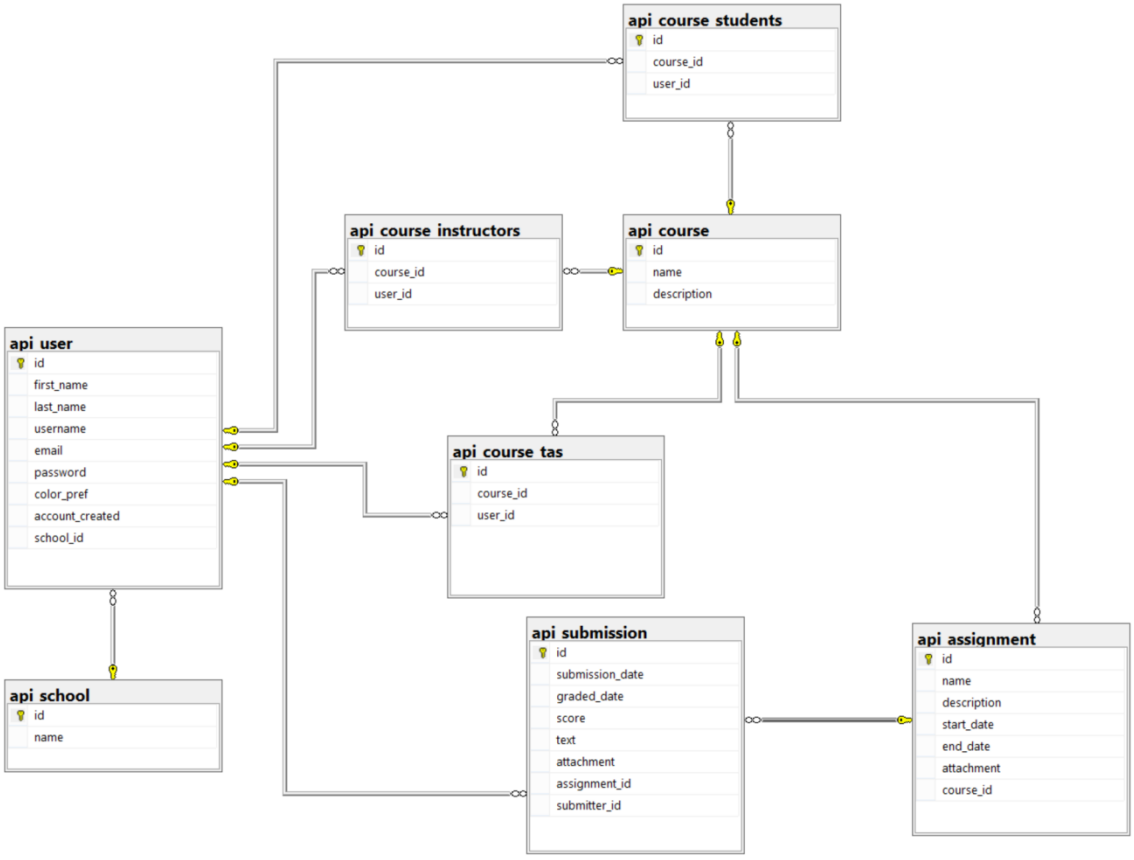

Figure 4 ER diagram of GME

4.2. Top Down Architecture

4.2.1. Architecture overview

GME system uses Client-Server architecture pattern. It consists two parts: the front end and server end. The frontend uses a NERD stack and communicates with the backend through axios API calls. The backend runs django and is written in Python, and communicates with an Azure SQL DB. The backend creates a website that can be used to view the data in the database easily, and also creates a variety of API calls that the frontend can use.

All dependencies of our frontend is handled by npm, including the automatic selenium tests.

4.2.2. Frontend Design and Framework

Our frontend is using REACT library to create interactive UIs and communicates with the backend through API calls. Frontend is implemented in a component-based way, in which all logic is written in JSX, which provides a way that uses syntax familiar to many developers, and all states are kept out of the DOM. The React library use abstract copy, the virtual DOM. This isolates all interfaces from any changes

or updates applied by users, and makes updates quick. REAC also allow us to create abstracted and reusable code component of different level, which makes the developing process more efficient.

Axios API call is used to communicate with backend APIs. For example, to make a user authentication backend API call:

```
axios.post(_CONFIG.devURL + '/user/authenticate/', formData)
```

Where `_CONFIG.devURL` is the backend API root, and `formData` is the input user credential, in our project we use email and password to authenticate user.

Design

The following diagram shows the structure of major parts of our frontend:

```
components/
├── Assignment
├── Class
├── _config
├── Course
├── Dashboard
├── Grade
├── GradeCenter
├── Home
├── Login
├── Origin
├── Register
└── Setting
```

Frontend organize files in folders that grouped by feature. Each component has everything it needs to work on its own, such as .scss, .jsx, which makes it reusable anywhere else. For example:

```
├── Dashboard
│   ├── Dashboard.css
│   ├── Dashboard.css.map
│   ├── Dashboard.jsx
│   └── Dashboard.scss
```

Also, grouping by features make things easier to work with, since all related files are grouped together. This at the same time saves us a lot of time.

Testing

We are using Selenium framework to do the frontend testing. At first, we used python to write the tests, however, it cannot be handled by npm. This forces us to switch to JavaScript. In addition we use Mocha framework to handle asynchronous testing and all dependencies are handled in npm.

Since our testing image does not support window system, in order to set up the CI build, we are using headless testing, in contrast to the end to end testing.

4.2.3. Backend Design and Framework

Our backend is using django framework and communicates with our Azure SQL server whenever objects are modified. Our SQL server was deployed first to azure (originally being a local django lite database), then our backend.

The expected way we implemented our database objects and the way we actually implemented them were very different, due to the way django operates. Originally, we had designed a SQL database structure for handling all the objects, including intermediate tables for handling many-to-many relations (such as, students and classes, since a class can have multiple students, and a student can be in multiple classes). However, django can handle this all behind the scenes. So, we would design the objects in Python, and created attributes for whatever we wanted the database to store. Then, django would manage the communication with the server silently. For example, if "c" is a course, c.students would quietly, behind the scenes, load all the students in course c and return them as python objects. This makes development and testing much simpler.

Example code:

```
    def get_users(self, request, pk=None):
        """
        Get list of assignments of a course with the pk
        :param request: it has to be GET
        :param pk: primary key of a course
        :return: assignments of course with <pk>
        """
        course = self.get_object()
        tas = course.tas
        students = course.students
        instructors = course.instructors

        response = {
            'tas': UserSerializer(tas, many=True).data,
            'students': UserSerializer(students, many=True).data,
            'instructors': UserSerializer(instructors, many=True).data,
        }
        return Response(response)
```

Automatic testing of backend uses the test suites provided by django test-execution framework.

Backend API:

Location: UserViewSet.assignments_due_week

How to use: Get request to /user/id/assignments_due_week/
Description: Gets all the assignment due within the next week for the user with user id. Returns a list of assignment serialized objects.

Location: UserViewSet.get_course

How to use: Get request to /user/id/get_course/
Description: Gets all the courses that a user is a student of, ta of, and instructor of. Maps 'student' to a list of courses, 'ta' to a list of courses, and 'instructor' to a list of courses.

Location: UserViewSet.authenticate

How to use: Post request to /user/authenticate/ with 'email' and 'password'
Description: Post the email and password of the user. Checks to see if they are valid (returns 'No such user' or 'Invalid password' error), and returns the user's first name and a token encoding their id and email.

Location: UserViewSet.encode_token

How to use: Post request to /user/encode_token/ with whatever data to store in token
Description: Post any body, and returns with a jwt encoded token of the data.

Location: UserViewSet.decode_token
How to use: Post request to /user/decode_token/ with 'token'
Description: Returns the decoded token. If incorrectly signed, returns 'invalid token'

Location: UserViewSet.create
How to use: Post request to /user/ with email, first_name, last_name, password, color_pref, and school
Description: Creates a user. Overrides the default create to hash the password.

Location: UserViewSet.change_password
How to use: Put request to /user/change_password/ with email, old_password, new_password
Description: Changes the password of user email from old to new password. Returns 201 on success, and 'No such user' or 'Invalid password' if old_password is incorrect.

Location: CourseViewSet.get_assignments
How to use: Get request to /course/id/get_assignments/
Description: Gets all the assignments for course with id

Location: SubmissionViewSet.get_submissions
How to use: Post request to /submission/get_submissions/ with userid and/or assignmentid or neither.
Description: Gets all submissions filtered by one, both, or none of userid and assignmentid.

Location: CourseViewSet.get_users
How to use: Get request to /course/<id>/get_users/
Description: Gets all users (tas, instructors, users) registered with courseid.

Location: CourseViewSet.save_course
How to use: Post request to /course/<id>/save_course/ with course_name (string), course_description (string), students (string of emails), tas (string of emails), and regenerate_access (bool). All of these are optional, and if not passed, nothing will occur.
Description: Updates the course name and description. Finds all new and removed students and tas and remove them. If regenerate_access, recreates access codes.

Location: UserViewSet.join_course
How to use: Post request to /user/<id>/join_course/ with access_code (string)
Description: Adds the user to the course given by access_code (also encodes student/ta), or returns an error if invalid.

5. Reflections

Sam Stephens (scsteph2):
I think our team was in a very unique position due to our small team size. We had the smallest team at 5 people, and one of those was an online student so it was very difficult for us to implement everything that we expected to. This made some aspects of XP take a new form, since something like collective code ownership is so necessary and in some ways hard to avoid. Additionally, essentially all features that

we had to implement were “core” features since our original task was so large relative to our size. Especially in the last couple iterations, the priority seemed to be very focused on getting a basic working implementation, rather than a sophisticated or detailed one - something also suggested by our TA. Our final website actually seems complete and looks good. It doesn’t have regrade requests, or advanced student analytics like I would like, but in many ways that made it a more educational project. Almost every feature required some large step in understanding. Learning how to use the django API testing framework was a larger portion of my time than “just” writing code, compared to if someone else could also help with that. Unfortunately, this also meant that the final solution of code was small. Twice the number of students would have definitely translated to much more than twice the amount of features and code, due to that overhead. The primary thing I learned is Django. I feel like I have a very strong understanding of how our backend system works and how I would expand it. Since Eric switched teams, I was essentially the only person doing any work on it, and unfortunately did not learn as much about the frontend as I would have liked.

Our process seemed effective. It was very similar to what our team did in CS427 as well. We have a few team meetings between each TA demo, where we work together for a few hours and catch up. In-between these meetings we do some of our own work and coordinate through Slack. Personally, this is a really nice balance between being able to work alone and being able to work with a team. Working alone is typically more effective if you know what you’re doing, working with a team helps work out specifications and design. We also gained a much stronger sense of how we could each be most effective, and this created a really good feeling of flow and understanding near the end.

Xin Cheng (xcheng11)

The primary thing that I have learned from this project is the Extreme Programming. The software projects that I did before use Scrum, which focus more on design instead of engineering practices. Compared to Scrum, XP is more suitable to our needs, since our team is made up of developers who are good at programming practices.

Team working is especially important and it helps us complete our tasks as well as keep everyone on the same page. Feedbacks from team members in team meetings and from TA in TA meetings help me understand the specifications more.

All of the frameworks used in this project are new to me and I am happy that have learned so many new skills at the end of this project. After iteration 4, I work primarily on the frontend testing. Since our previous testing didn’t work, I have to start from scratch. Fortunately, I am able to set up all local environments and make the testing work. With the help from other team members, we finally set up the CI pipeline for frontend. While writing the test cases for frontend, I become more familiar with the React library, plus the previous developing experience in backend, I have learned almost all the frameworks used in this project.

Overall, I think our process is successful and effective. With the help from each other and TA, we have implemented almost all the essential features, and our websites works well.

Alvin Chang (ahchang6):

I was a little disappointed with the size of the team because the instructor had intended for teams to be around 8, but our team had essentially 5 people including an online student. I think it brings a different set of challenges that was envisioned for this class; instead of planning and collaborate problems, we had a limit on the progress of our project due the team size.

One challenge I had encountered was from the difference in experiences of frontend.

This difference in experience includes teaching React and teaching frontend development. Before this, I was the only with a significant amount of experience in frontend/React, this led to limitation of productivity and implementation of features. Since I not only had to implement the more "tricky" features, I still had to guide through the frontend team by teaching basics about React and fix bugs that they encountered along the way. Majority of this teaching was towards the beginning of the project, and this was lower towards the end of the project.

Overall, I felt more invested into this project than CS427 because we had developed the project from the bottom up, and we felt accountable for every bug that we made so we tend to fix them quickly, efficiently, and without complaints.

I was a little unsatisfied with our overall progress and how much we got done out of the planned user stories. From the beginning, we underestimated the difficulty of learning a framework and applying it. This makes me wonder if project assignments should focus on student skill sets more since time availability tends to lean towards unwillingness vs unable.

I think I did learn the React framework better because I had to teach it often and I had completely developed in React this whole project. I was interested in learning another framework but it seems no one knew any other framework, so we ended up choosing React because I knew it well and we felt that everyone learning a new framework would lead to challenges that we could not imagine.

I both enjoyed the process because I thought it was clear and effective. We had multiple meetings per week and discussed the problems we had and communicated some additional requirements we needed from the other team. Since majority of the beginning, I was guiding others in React, partner programming was an essential part of this process. It was important for them to try it themself and I would offer feedback if there were better ways. I think each of us had a good sense about each of our strengths and weaknesses so we assigned requirements based on the strengths and still read code written by others regarding our weaknesses.

Jozef Cibicek (cibicek2)

Picking up and learning React right from the start seemed like a big challenge but with the right dedication to learn and willingness to help from other team members made it overall a wonderful experience. The sheer size of the team made it little frustrating in the beginning, but it was an advantage from my point of view later on because we had less lines of communication in our team and thus we were more flexible to meet and discuss issues. Since this was my first software project as a member of frontend, I got to learn numerous aspects of frontend development such as working with requests, API's,

and React. I also worked on the user interface elements of the project which made me to utilize my knowledge learnt for the first time in a project of a larger size. We could not implement several features that I would like to see on the web page, as the other members mentioned, due to the smaller size of the team which made me little disappointed.

SeokHyun Song (ssong11)

I learned lots of technical skills from this project including Azure, React and Django rest framework. Learning Azure was one of the challenges for me since I didn't have a chance to use it. There were many resources that I could use such as database, server and deployment. However, I had to figure out how to use the resources. For example, to deploy our backend product to Azure, I had to learn and create custom web config and deployment config file because default web config and deployment config didn't support Django app. Learning React and Django restful framework were not that hard for me since I had an experience with Javascript and Django application.

From this project, I realized the importance of refactoring. We had to do refactoring at the end of every iterations since our team decided to follow XP process. Before doing refactoring, our code was not readable and it was extremely hard to debug. Although refactoring takes time, it helped us to find bugs, improve frontend design, and make codes readable.