CSCB20 Assignment 3 Report

Ayaan Shahab
Harguntas Benipal
Quincy Lee

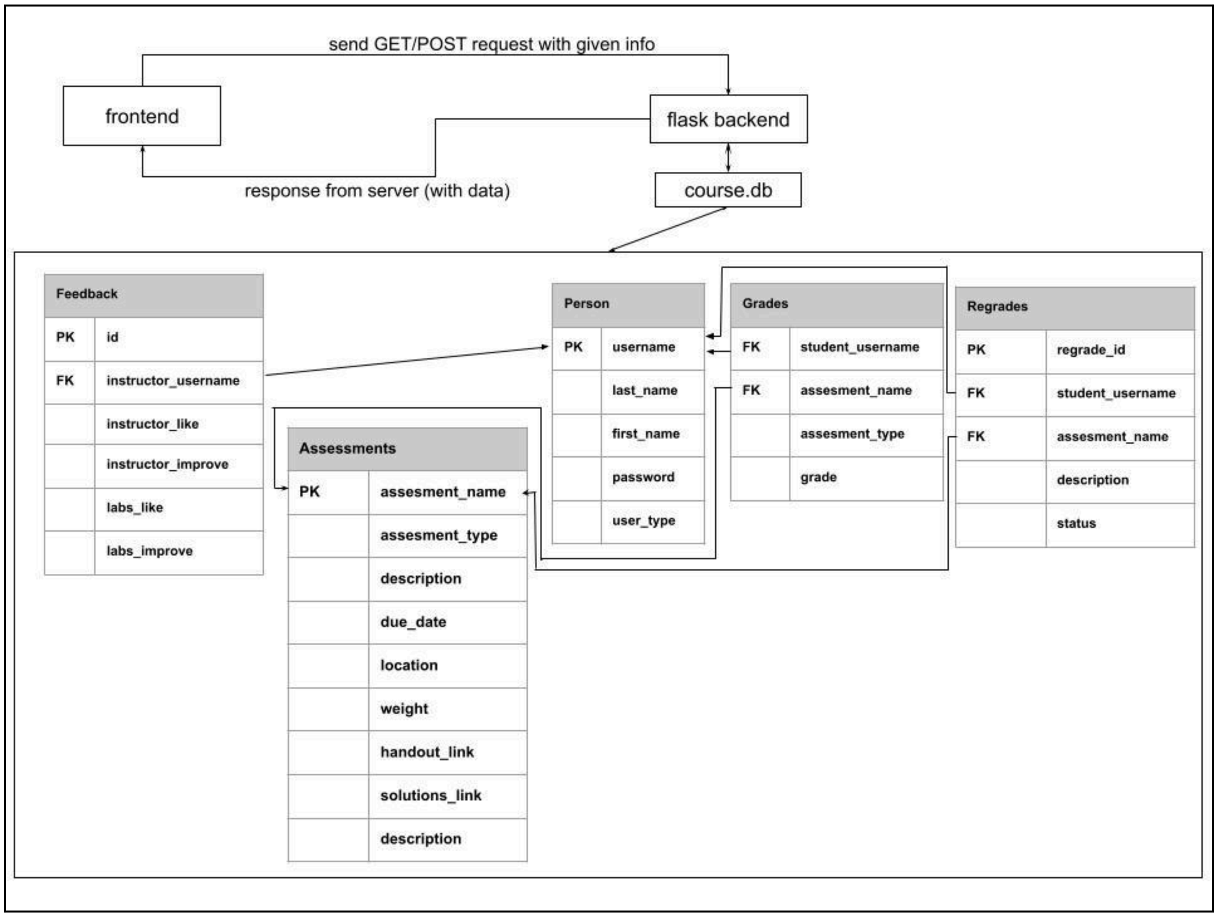
April 6, 2024

<u>Note</u> that the .zip file may contain a .DS_Store directory and files similar to that. Please ignore these files and directories as they are included by the operating system when zipping files.

Integrated Features

The main features we integrated on Assignment 3 pertain to the creation of a fully functional backend. This backend consists of a Flask Server, meant to render files and handle all interactions with our database. These interactions were done through the use of Flask-SQLAlchemy.

1. Database Schema Design



As shown in the diagram above, we structured our database (course.db) into 5 relations. These 5 schema's are: Person, Grades, Regrades, Assessments and Feedback. We used concepts such as foreign key relationships to allow for efficient querying and easy use. Furthermore, we used Bcrypt to hash passwords in the Person relation.

2. Frontend:

- We made the header navigation bar more reactive and appealing by adding transition CSS effects
- We used Javascript to have our header bar resize according to the users screen size
- Since this website was meant for both students and instructors, we set up authentication checks on all of our pages, and redirected users that were not logged in, back to the login page
- We also made sure through server side validation that users are not able to access pages that they are not authorized to by changing the url in their browser
- Furthermore, we had different pages for students and instructors, as discussed below:

Student and Instructor:

We created a personalized page for each user called "My Account". The functionalities of this page depended on whether the user was a student or instructor.

Students:

For students, this page provided access to their **current grades**, allowed them to submit **regrade requests**, and give **feedback** to their instructor. There was also a logout button for students.

Students could request a regrade for each assessment once from their grades page. This was done by clicking a button that directed them to the regrade request page, where they could select the assessment and provide a description of their reasoning.

Instructors:

For instructors, the page allowed them to view **student grades**, add **new grades** and **new assessments**, manage pending **regrade requests**, review **feedback** and **logout**.

When instructors resolved regrade requests, they could view the current grade, student's name, description, and input a new grade along with feedback.

HTML Functionality:

We adapted our web design from a previous assignment, incorporating Jinja templates for HTML. We utilized Jinja's block content feature to link different pages of our website.

Additionally, we implemented a homepage with login prompts and a course description for logged-in users. An error page was added to handle invalid URLs, redirecting users to an error page. Login and register pages were included for both students and instructors.

For instructors, we added:

A page to add assessments
A page to add student grades
A page to do regrades
A page to view student feedback
A page to view student grades
A page to register (as an instructor)
A page to resolve and respond to students' regrade requests
A page to view students' regrade requests
A page to view the links to the different instructor specific webpages for easy access.

For students, we added:

A page to view their grades
A page to submit regrade requests
A page to view their regrade requests
A page to view the links to the different student specific webpages for easy access.

3. Backend Development

We employed SQLAlchemy with Flask to manage data storage for assessments, feedback, grades, and users (both instructors and students) in a SQL database. The database consists of tables for assessments, feedback, grades, person, and regrades.

To determine database actions, we distinguished between GET and POST requests. POST requests were used for adding data to the database, while GET requests were used to display pages. For instance, when a student requested the regrade form page using a GET request, we displayed the form.

Upon submission, using a POST request, the data was added to the database. This approach was consistently applied across all pages handling GET or POST requests.

Additionally, we utilized SQL-Alchemy queries to identify specific tuples. The provided app.py file exemplifies a Flask application utilizing SQL database operations, HTTP GET, and POST requests for various functionalities related to course management.

Inside app.py, we utilized helper functions and created parsing functions to read the date. This meant that we had to update the global variables "app.jinja_env.globals.update(parse_date=parse_date)" because the Jinja template needs to do so.

Lastly, we utilized comments in the app.py file for better readability and maintenance.

Features not yet completed

We could not implement a search function, enabling users to effortlessly locate specific content on the site. In the future, we hope to use frontend frameworks in Javascript to allow this to be possible.

We wanted to allow the upload of diverse file types, including images, text files, and videos, for use in lectures. Another useful feature we could not implement is a HTML/CSS validator built into the website.

Lastly, another feature that we could not yet complete is making our signup/login service more secure by sending confirmation emails to users. This could significantly lower the amount of bot accounts that are made, leading to less latency in our database and better experience for actual users.

Sections worked on by group members

Ayaan Shahab:

- Implemented "My Account" UI and buttons

- Converted all HTML files from assignment 2 into Jinja Format
- Created add_grade, regrade, feedback, etc functions in app.py
- Created authentication checking in functions
- Wrote tests to check for correct format of data in POST requests
- Connected backend with frontend with GET/POST requests
- Implemented Calendar UI and time UI pages

Harguntas Benipal:
- Wrote final report
- Created logical schema for database (see image provided above)
- Implemented view regrade pages for students and instructors
- Created adjust.js file to allow header to be resizable
- Linked all image/example images/pdfs in html code
- Created parsing function to parse date into correct format
- Implemented UI to allow students to see finished regrade requests and ones that are still pending

Quincy Lee:
- Implemented database logic in app.py using SQL-Alchemy
- Created database queries to insert and uniquely find tuples in course.db
- Made CSS files modular by splitting up files into useful files (no repeated code)
- Created instance of course.db and added starter data for marking

Percentage Distribution of Work

The sections worked on by each group member are stated above.

Ayaan Shahab: 33%

Harguntas Benipal: 33%

Quincy Lee: 33%

Final Thoughts

Overall, we found that we were able to improve substantially on previous websites by using a better design through flexbox, grid, and media queries so that the website looked not only visually appealing, but also looked correct on every device's screen width.