Project Final Report for Chinmay Arvind - TEAM NUMBER 8 (AIvaluate)

First and Last name: Chinmay Arvind (Student ID: 81088635)

Team Members: Chinmay Arvind, Colton Palfrey, Jerry Fan, Omar Hemed, Aayush Chaudhary.

Overview

My team consists of 5 members in total including myself, they are: Colton Palfrey, Jerry Fan, Omar Hemed, and Aayush Chaudhary. Our client is a recent graduate from Simon Fraser University with a Master of Science in Computer Science, his name is Parsa Rajabi. The goal for us as a team during this project was to produce an accurate, powerful, and efficient software implementation of an AI-powered web development course platform. Our approach for this project was to allow for students to join courses, submit their assignment solutions (as CSS, HTML, or JavaScript files) for instructors to be able to access, and view their final grades and feedback released by the instructor with the help of a text-generating AI model. The instructors will be able to create courses for students to enroll themselves in, access the students' submissions with a click of a button, automatically grade students' submissions, and generate valid feedback with a grade based on a rubric and the instructor's style of grading. These AI-generated grades and feedback can serve as a guide for the instructor's final grade and feedback of the students' submissions which can be released for students to view. Administrators of our application have the ability to manage courses, student enrollments in courses, instructor-level access for courses, and possess the ability to manage account details for students and instructors. Our project's purpose was to meet the following key system requirements:

Functional Requirements

- System will render a neat user interface built based on the technology stack mentioned later in this document and the UI designs established in the design plan document in our code repository for this project.
- System will render a login and signup page for all types of user roles: students, instructors, and administrators.
- System will check credentials of a user that is logging in against a database that stores the user details and creates new credentials for users that are signing up to be stored in the database for each type of user other than the instructor (the administrator handles the creation of instructor accounts).
- System will redirect to different dashboards (or home pages) based on the role of the user that is logging in (students, instructors, or administrators) using a reverse-proxy that ensures users can only see on their screen what is meant to be rendered for their account's user type.
- System will allow only users with the user role of an instructor to be able to create, read, delete, and update assignments based on their instructor ID which is stored in a database and can be confirmed.
- System will allow only users with the user role of an instructor to be able to view, create, and update rubrics based on their instructor ID and the course's ID, which is stored in a database and can be confirmed.
- System will allow only administrators to change the access privileges of instructors from being a professor to a teaching assistant and vice versa giving teaching assistants the ability to assist with the grading process, but not handle course-level functionalities like editing a course's details, archiving a course, deleting a course, assigning or unassigning teaching assistants from a course, and testing the AI model before using it for grading student submissions.
- System will allow only users with the user role of an instructor (includes teaching assistants) to be able to view and mark student assignment submissions with the help of AI-generated feedback and grades from an API-based Large Language Model (LLM) response, which are stored in a database holding grades, feedback, student information, instructor information, enrollment information, assignment information, assignment submission information, rubric information, and AI prompting information that are all tables or fields in table linked to each other via foreign keys.
- System will only allow for all student submissions to be graded by an AI model (LLM) automatically with the click of a button on a page from the user role of an instructor (teaching assistants included), using the help of the assignment's rubric and the AI prompt that an instructor can add to their message to the LLM when grading student submissions to specify subjective or objective factors for grading submissions.

- System will allow for only users with the role of an instructor to be able to view and release final grades and feedback for all students and allow for users with role of a student to be able to view the released feedback and grades, which are all retrieved from the database storing the information related to feedback and grades.
- System will pull information from the database holding the assignment information and render it for both user roles of students and instructors for the user roles to efficiently perform the actions of submitting assignment files and grading assignment submissions respectively.

Non-functional requirements

- System will have privacy measures in place for processing student submissions in a manner that does not disclose students' personal information to the LLM to ensure unbiased AI grading through naming storage directories for submission files in a purposefully ambiguous way (explained in the Features Individually Completed section of this document) and storing the data in a renowned relational database management system like PostgreSQL to prevent foreign parties from accessing sensitive student data.
- System's frontend will be developed using React.js, and the AI component will be developed using OpenAI's Assistants API.
- System will be containerized using Docker containers to ensure that the code for this software can be run on any operating system.
- System will not share files uploaded by students and instructors across devices running the same code and docker compose file because the storage of files on docker volumes is local to the device that the code is being run on and has been setup in this way to meet our client's requirements of avoiding file storage on cloud-based solutions.
- System will handle a maximum of 7 solution submission files (with reuploads allowed) and a GitHub Gist link, and a single answer key file from the instructor at a time in the accepted file formats defined above (with reuploads allowed).
- System will use an instructor-provided prompt (optional) for subjective or objective measures for grading (efficiency, correctness, etc.) and an instructor-provided rubric to produce sensible and actionable AI-generated feedback and grades using the Assistants API within 2 minutes for an individual student's submission files.
- System will be logged into while running the automatic grading of assignments with the AI model and will halt grading as soon as an instructor logs out of the application.
- System will have an intuitive user interface built with React.js and consisting of icons, text, and tooltips to support straightforward navigation between different parts of the application for all user roles.
- System will be written in a way that makes the code-handoff process smooth using JavaScript comments describing each component of the code.

User Requirements

- An instructor will be able to create assignments with a rubric for grading assignment submissions, view assignment details, and update assignments by changing the due date or reuploading an assignment key, as well as being able to delete assignments that they create.
- An instructor will be able to view and update rubrics for an assignment by changing the text in a rubric.
- An instructor (including teaching assistants) will be able to view, and edit feedback generated by the AI, as well as override the grade generated by the AI through providing an instructor-given final grade with feedback individually for all the students whose submissions were graded by the AI before making the grades and feedback public for the students to view.
- An instructor will be able to set a prompt that is sent to the AI to match their style of grading (to grade submissions based on correctness, grade submissions strictly based on some criteria, etc.)
- An instructor will be able to recycle rubrics created by them for any course when creating new assignments.
- An instructor with teaching assistant-level access will not be permitted to access course management functions like being able to archive a course, delete a course, assigning or unassigning teaching assistants from a course, and testing the AI model before using it for grading student submissions, but can provide feedback and grade submissions.
- An instructor will be able to view the entire course's grade-wise and feedback-wise performance on an assignment, as well as a class-wise final grade average.

- A student will be able to upload .html, .css, .js, and .jsx submission files from their device for an assignment and will be notified whether the submission files were uploaded or not (in the case of an error).
- A student will be able to see their average mark in the course, and their final instructor-released marks with feedback on each individual assignment.
- A system administrator will be able to manage instructor, student, and course information – specifically by having full-access to enrollments, course details, and user account information aside from passwords.

Technical requirements

- System will be architected so as to not use a cloud-based storage solution for assignment keys and submission files following client feedback on data privacy laws.
- System will be architected to accommodate many users without any issues as file storage is not shared across a network and will be stored locally using docker volumes.
- System will be architected in a manner that encapsulates functionality for user roles within docker containers, allowing for cross-platform ability to run the application, and increasing the number of points of failure ensuring that even if one service goes down, the rest can continue working.
- System will be architected in a manner that allows for an instructor to be able to send a specific prompt with their unique grading style to the OpenAI's Assistants API (Application Programming Interface) along with the answer key and the rubric for an assignment ensuring that the grading of students submissions happens with the instructor's vision in mind.
- System will be architected in a manner that allows for past-created rubrics to be used again across courses, and restoration of deleted users and courses from an administrator user role's perspective.
- System will be designed in a manner that supports restriction from certain resources based on user roles (student, administrator, instructor), and will also be implemented with a CI (Continuous Integration)/CD (Continuous Deployment) pipeline which will directly test and ensure all tests pass before any new features are merged into a production environment.
- System will be architected in such a way that unit testing will be conducted with all features, and user acceptance testing will be conducted during testing sessions to gain feedback and improve the product.

Unique Value Proposition (UVP)

The Unique Value Proposition of our project is to give instructors the means to create and manage courses as they see fit and use the power of AI to grade student submissions automatically. The grading process is initiated with the creation of assignment with an appropriate rubric, along with the creation of prompt that describes subjective or objective measures for how the AI should generally go about the process of grading the students' submissions for the assignment at hand. The AI aims to mimic and apply the grading style of the professor to the students' submissions. The AI model receives the answer key (if provided), an AI prompt on how to go about grading the submissions, and the parsed content from the student's submissions (both GitHub Gist links and submission files in .html, .css, or .js formats) as part of the grading process. The AI API generates a cohesive response in a format that can be parsed, allowing for grade and feedback extraction, which is then displayed to the instructor for every student's submissions. These automatically generated grades and feedback can be used by the instructors and teaching assistants as a starting point for their evaluation of the student's performance and improve final feedback by providing insights instructors would have otherwise missed if they chose to manually grade submissions (three sources of feedback: instructor, teaching assistant, and AI). Allowing only the instructors and teaching assistants to see AI feedback is done in order to maintain full human oversight of AI-generated feedback and grades to weed out nonsensical feedback and grades in the case that the AI model fails to grade a submission, providing highly vetted feedback that can lead to students' identification of problem areas for future assignments in the course.

System Architecture

Final architecture

The final architecture of our AI-powered web development course platform follows a microservices architecture. The encapsulation of different major components of our application that deal with different use cases of the system are

containerized within separate containers, allowing for the cohesion to be high within major components of our system and inter-container communication happening through the use of a docker network. The advantage of having a microservices architecture is to increase the amount of failure points in the system, ensuring that even if one or more of the services goes down due to any range of errors, the rest of the system runs independent from the affected containers due to the coupling between the system's modules being loose, allowing users to continue using the application's unaffected modules.

Final Technology Stack and Justifications Discussion

The final technology stack that we decided upon as a team to execute this project's implementation is as follows:

- Node.js: Facilitates general backend development and running of server code in a common language like JavaScript and is highly compatible with React.js as they are both written in the same language. Since our client expected our product to be built with a React.js frontend, we decided it was best to use Node.js as our backend base.
- Express.js: Facilitates the creation of routes for the web application to update, delete, insert, and get resources from our docker containers running other components of our system, while managing session variables (set in the backend to perform functions such as: checking whether a user is logged in or not using their ID, and whether course details have been set or not, etc.). Since we decided our backend would be built with Node.js, we decided it was best to use Express.js for the bulk of our backend development due to its high compatibility with Node.js.
- React.js: Facilitates advanced frontend development to render webpages as specified in our design plan, the ability to access and modify data in the database through Axios, and communication with routes controlled by Express.js in the backend containers. It is highly compatible with Node.js and Express.js backend code and was recommended by our client.
- Passport.js: Facilitates user authentication by working with Express.js to ensure only authenticated users are able to login and access all resources for their respective user roles. Passport.js was chosen as our preferred authentication framework due to its high compatibility and common usage with Express.js-based backend applications.
- Nginx: Facilitates reverse-proxy functions by requesting the user role from the client side, and accurately routing the user's frontend to be rendered based on their user role and only allowing them access to the webpages and send requests to backend endpoints defined for their respective user role. Nginx was used as the preferred choice for the reverse-proxy due to its ease of integration with the frontend and backend components over our previous choice (http-proxy-middleware).
- Docker: Facilitates the running of cross-platform developed code with the use of docker images, containers, and running instructions within docker files to download required packages for our software to run without any issues. This allows our application code to be built and run independent of the operating system that the user is running the application on, as well as handles the splitting of the application into microservices that communicate requests with one another via a docker network. Another advantage of using Docker was also that it had container-wise local storage in units called volumes which we used for file storage of assignment submissions for students, and answer keys for instructors. The use of Docker was strongly suggested to us to build a scalable microservices-based system.
- PostgreSQL: Facilitates relational database management by storing data in tables that are related to each other via foreign keys, and efficient querying as well as modifying of data from all backend routes defined using Node.js and Express.js backend code. PostgreSQL was our preferred choice for an RDBMS (Relational Database Management System) due to its efficiency over the RDBMS that most of our team members' had prior experience with (MySQL) and due to its similarity in syntax to MySQL.
- Drone CI: Facilitates the creation of a pipeline for continuous feature integration that runs the tests scripts developed with Jest for unit testing before allowing for a pull request with new features on it to be merged into our development branch, and further into the master branch. Drone CI was suggested to us due to its compatibility with GitHub, and therefore was used by us in our project.
- Jest: Facilitates creation of unit tests to test our individual files of code running backend routes, as well as frontend pages. Jest is the most popular unit-testing framework in JavaScript, and we decided to use it as some of our team members had prior experience with testing using Jest.

- VITE: Facilitates the creation of a React.js application with basic requirements of a web application satisfied and pre-installed, which we used as a starting point for our development. VITE was used by our team over create react app as create react app was being deprecated, and using a framework like this reduced our work of having to figure out how to build everything in our web application from scratch, which we were instructed at the beginning of the course to avoid doing due to time constraints of this project.
- OpenAI Assistants API: Facilitates generation of text responses containing feedback and grades based on a provided thread, run, message, files, rubric, and assignment key (if provided) which is then relayed back to the frontend for the instructors to view the AI generated grades and feedback. Our team chose this API for our AI component as it produced faster and more relevant feedback over the open-source Ollama models like Llama-3 (8 Billion parameters) and Code-Llama.

Major components of system

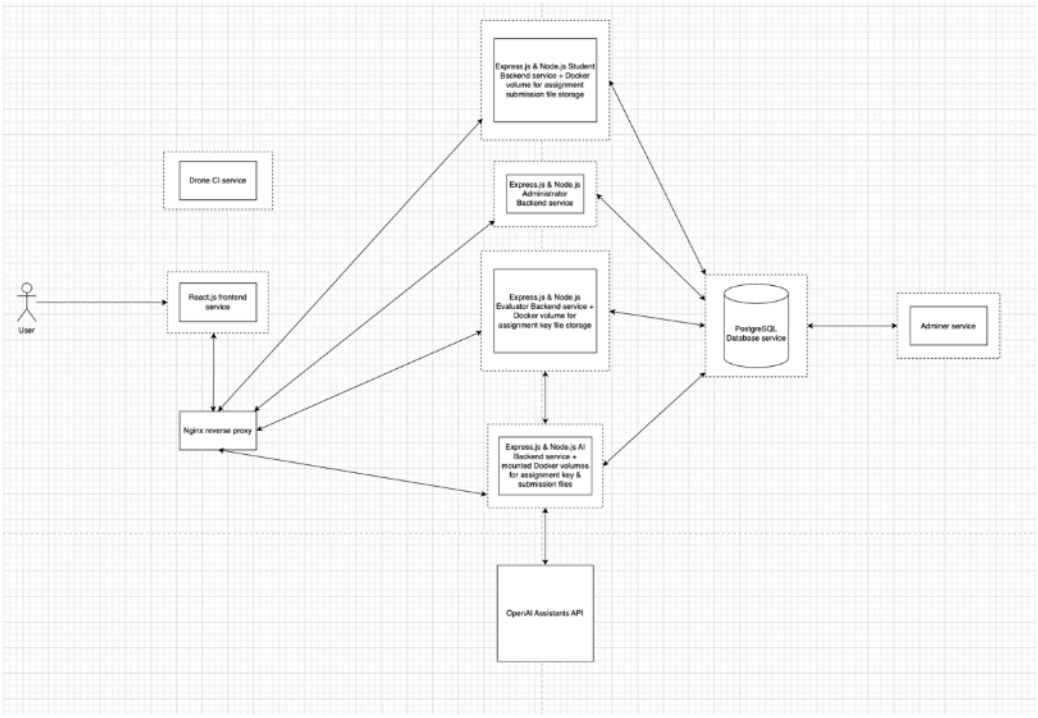
Our web application consists of the following major components:

1. Reverse proxy to route users based on their user roles (student, administrator, instructor) to be able to access web pages meant for their user role by requesting the frontend for the user role and sending it to the reverse proxy set up in the nginx.conf file (located in the frontend directory of our project), where the appropriate backend container is routed to allowing for conditional rendering of frontend pages for all the user roles.
2. Frontend container to encapsulate all the functionality and user interface code for all the web pages that can be accessed by all user roles, with a docker file containing installation instructions for appropriate packages. This is dependent on the student backend, administrator backend, and evaluator backend containers, and is part of the network connecting all the containers so that they can communicate requests with one another.
3. Database container to encapsulate all the data within the database, built on the PostgreSQL docker image and uses the database connection credentials established throughout various .env files in our application's backend code and runs in collaboration with the dbCofig.js files that create the connection to the database. This container has a volume attached to it to store relevant data generated within the container and is part of the network connecting all containers to enable communication between the backend and the database.
4. Adminer container to encapsulate the user interface and database managing functionality for the adminer UI to access and modify the data within our PostgreSQL database and ensure that the data is being accurately stored to aid our development. This container is also a part of the network connecting all containers.
5. Student backend container (port 4000) to encapsulate all backend routes and functionality that deal with accessing or modifying student accounts, student assignments, student enrollments in courses, and student grades. The frontend pages that are visible to a user with the user role of student communicate with this container via requests. This container has a docker volume mounted on it to hold the student submission files based on a directory structure designed with the help of the student ID, course ID, and assignment ID. This container is dependent on the database container and has a docker file containing installation instructions for appropriate packages so that the application can run without errors on a user's machine. The .env file for this container contains credentials to access the database and perform CRUD (Create Read Update Delete) operations on the data present in the database container, which are then reflected by the adminer container and can be modified from Adminer's UI when opening a browser and searching up localhost with the port number 8080. The inter-container communication is facilitated by the docker network called aivaluate-net and has been configured appropriately.
6. Administrator backend container (port 3000) to encapsulate all backend routes and functionality that deal with accessing or modifying administrator account details, course details (along with restored courses), instructor details, student details, enrollment details of students, and teaching assistant-level access or instructor-level access to teach courses. The frontend pages that are visible to a user with the user role of administrator communicate with this container via requests. This container is dependent on the database container and has a docker file containing installation instructions for appropriate packages so that the application can run without errors on a user's machine. The .env file for this container contains credentials to access the database and perform CRUD (Create Read Update Delete) operations on the data present in the database container, which are then reflected by the adminer container and can be modified from Adminer's UI when opening a browser and searching up localhost with the port number 8080. The inter-container communication is facilitated by the docker network called aivaluate-net and has been configured appropriately.

7. Evaluator backend container (port 6000) to encapsulate all backend routes and functionality that deal with accessing or modifying instructor accounts details, student assignments, student grades, student submissions, AI prompts, and marking assignments. The frontend pages that are visible to a user with the user role of evaluator (instructor) communicate with this container via requests. This container has two docker volumes mounted on it to hold the student submission files based on a directory structure designed with the help of the student ID, course ID, and assignment ID, as well as the answer key files uploaded by instructors based on a directory structure designed with the help of the course ID, instructor ID, and assignment ID. This container is dependent on the database container and has a docker file containing installation instructions for appropriate packages so that the application can run without errors on a user's machine. The .env file for this container contains credentials to access the database and perform CRUD (Create Read Update Delete) operations on the data present in the database container, which are then reflected by the adminer container and can be modified from Adminer's UI when opening a browser and searching up localhost with the port number 8080. The inter-container communication is facilitated by the docker network called aivaluate-net and has been configured appropriately.
8. AI backend container (port 9000) to encapsulate all AI API-related functionality and dealing with AI response generation to be displayed on the instructor's frontend. The frontend pages that are visible to a user with the user role of evaluator communicate with this container via requests. The feedback and grades generated by the AI are viewable via logging and through the adminer UI (the AI API itself is not containerized, and therefore, not a part of our system). This container has the submissions and answer key docker volumes mounted onto it with the same directory structures as defined above and sends received AI API responses back to the evaluator backend container.
9. Drone CI container to encapsulate the functionality of the CI-CD system with the use of a docker image.
10. Volumes hold assignment submission files for students and assignment key files for instructors and are mounted onto the appropriate containers for easy access and modification.

The web application is operationalized by simply opening up a terminal in your preferred choice of IDE (Integrated Development Environment), navigating to where the code for this project has been downloaded using the cd (change directory) command till you reach the target directory (assuming you have Docker desktop app installed and running), and running the following command: docker compose up --build.

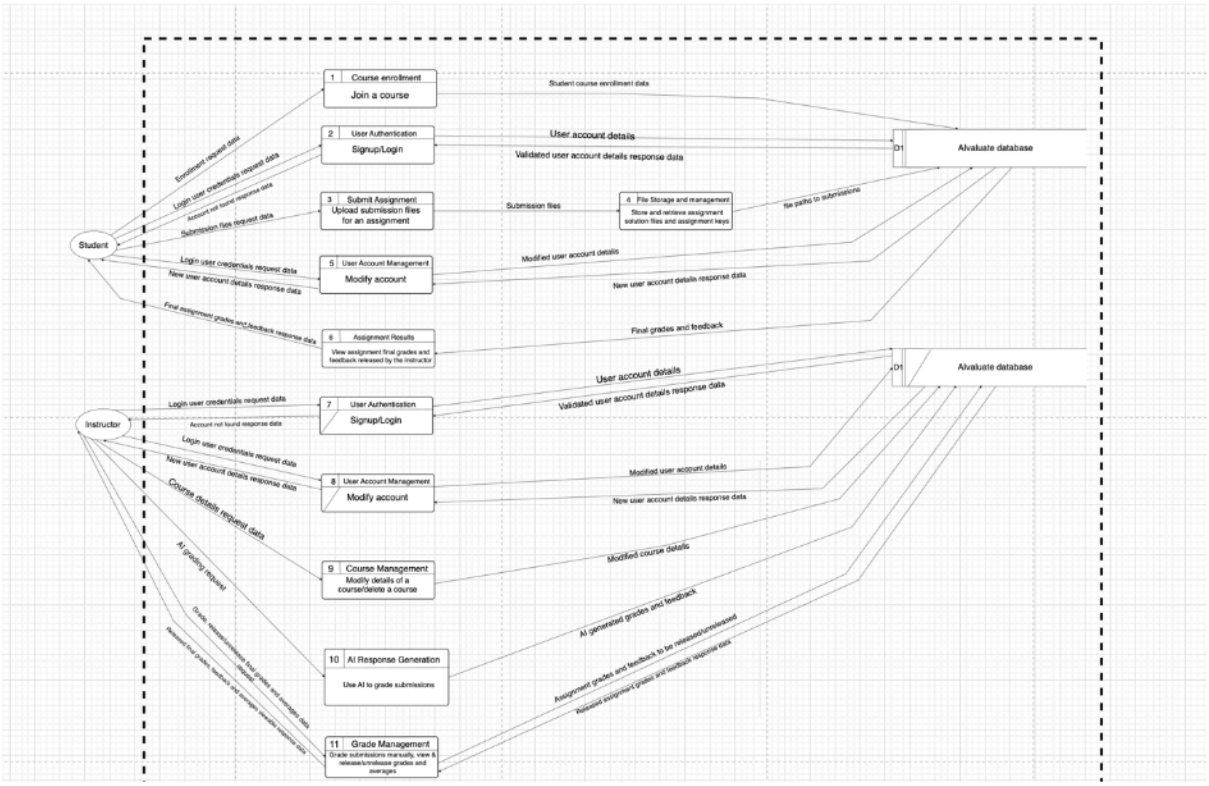
System Architecture Diagram (Zoom in to view diagram clearly)

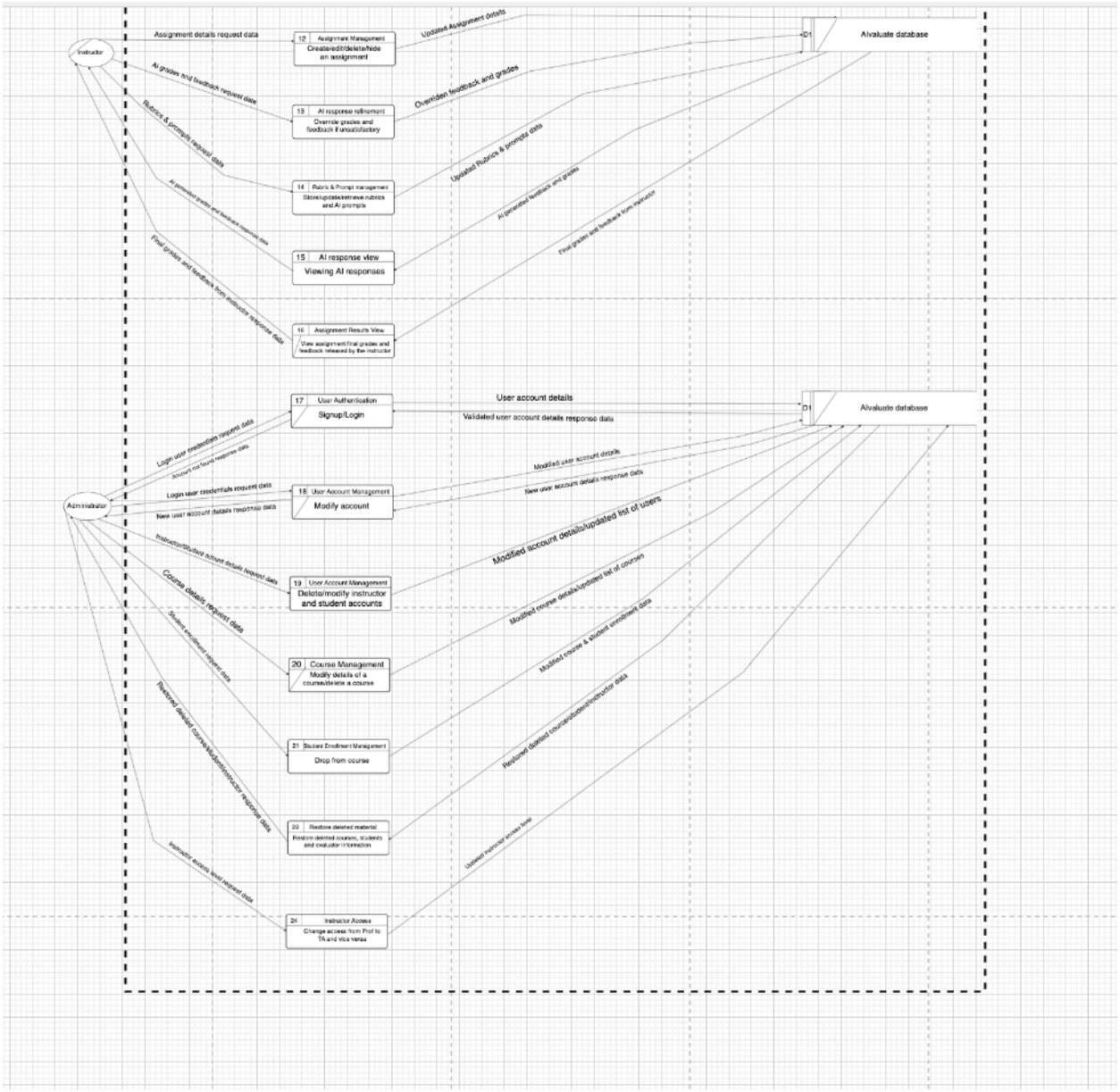


Explanation: The reverse proxy gets the user role from the frontend and routes the user to the pages controlled by the relevant backend container, from where, based on the user role, any of the following 3 scenarios can take place:

1. User role student: If the user is a student who is logged in or signing up, then they are displayed only student-accessible frontend pages which have backend calls to code contained in the backend student container containing requests made to access and modify data in the PostgreSQL database. File paths of assignment submissions are stored in the database under the AssignmentSubmission relation, and these file paths are the exact location of the students' submissions on the docker volume mounted onto the backend student container.
2. User role administrator: If the user is an administrator who is logged in or signing up, then they are displayed only administrator-accessible frontend pages which have backend calls to code contained in the backend administrator container containing requests made to access and modify data in the PostgreSQL database.
3. User role evaluator: If the user is an evaluator who is logged in or signing up, then they are displayed only evaluator-accessible frontend pages which have backend calls to code contained in the backend evaluator container containing requests made to access and modify data in the PostgreSQL database. File paths of assignment keys and assignment submissions are stored in the database under the Assignment and AssignmentSubmission tables respectively, and these file paths are the exact locations of the evaluators' uploaded assignment keys for their created assignments and students' uploaded assignment submissions on the docker volumes mounted onto the backend evaluator container. The evaluator backend container communicates with the AI backend container for the status of the grading, and the AI backend container communicates with the AI API for the grading and feedback of the submissions based on the rubric, answer key, and instructor-created AI prompt to guide the grading.

Data Flow Diagram: Level 1 (Zoom in to view pictures of diagram clearly) (Split into multiple pictures due to text becoming too small to view in diagram)





Explanation: The arrows connecting the oval shapes (external entities) to rectangles with key process names and a serial number (process boxes) represent the request data sent to trigger the process, and the arrows extending from the process boxes to the rectangle with the text D1 and AIvaluate Database (data store) represent data flow into the data store. There are also instances where data stores are used to retrieve data and flow back out of the data store into a process box, and when that process is performed, the data is processed and received on the external entities' side. The process boxes' functionality (within each process box below the title) for each key process give a hint as to the main data processing taking place for that user role and process, with the processed data flowing out to a data store before the data can be picked up by a different process for its key functions of retrieving data or using it for its own process. The external entities with a line passing through them are duplicate external entities (created for easier interpretability and viewability), and the process boxes with a line passing through them are duplicate process boxes, meaning multiple external entities have access to that same key process. The data stores with a line passing through them are duplicate data stores (created for easier interpretability and viewability).

Student users have access to key processes such as: joining courses, accessing assignments and submitting their submission files, signing up and logging in, modifying their account details (i.e. their password), and being able to

view the final feedback with grades provided on their assignment submission files by instructors. The request data for actions such as: joining a course, signing up and logging in, submitting their assignment submission files, and managing their account details is sent to the respective processes to trigger them, and once the data processing is completed by the respective process, the data generated is sent to the data store where it will be stored for all processes to use. There is one instance within these key processes for a student user, where the data store will have been populated with the required data for a process which is the process of viewing assignment results (feedback and grades written by the instructor based on the AI's feedback). This will be a data retrieval process from the data store, indicating the flow of data out from the data store through a process where processing is completed, and the data is visible to the appropriate external entity via the User Interface.

Instructor users have access to key processes such as: signing up and logging in, modifying their account details (i.e. their password), deleting courses or editing details of course (archiving a course, assigning or unassigning teaching assistants to or from a course, etc.), making use of the AI grading feature's responses for automating the process of grading student submissions; grading with viewing and releasing or unreleasing grades as well as averages, assigning final grades with instructor feedback for submissions by overriding AI-generated grades and feedback if it is inadequate; creating, updating and viewing rubrics as well as AI prompts to be sent to the AI model, and viewing AI responses containing feedback and grades for student submissions. The request data for actions such as: signing up and logging in, modifying their account details (i.e. their password), deleting courses or editing details of course (archiving a course, assigning or unassigning teaching assistants to or from a course, etc.), utilize the AI grading feature's responses for automating the process of grading student submissions, assigning final grades with instructor feedback for submissions by overriding AI-generated grades and feedback if it is inadequate; as well as creating and updating rubrics is sent to the respective processes to trigger them, and once the data processing is completed by the respective process, the data generated is sent to the data store where it will be stored for all processes to use. There are three instances within the key processes for an instructor user, where the data store will have been populated with the required data for these processes, which are: grading with viewing & releasing or unreleasing grades and averages, viewing rubrics as well as AI prompts to be sent to the AI model, and viewing AI responses containing feedback and grades for student submissions. These processes will be data retrieval processes from the data store, indicating the flow of data out from the data store through processes where processing is completed, and the data is visible to the appropriate external entity via the User Interface.

Administrator users have access to key processes such as: signing up and logging in, modifying their account details (i.e. their password), deleting or modifying students and evaluators accounts, deleting or editing details of courses (assigning instructors to a course), dropping students from courses; restoring deleted student, evaluator and course data and being able to view it instantly post-restore; and modifying instructor access to course management abilities by changing the role of an instructor to a teaching assistant or vice versa. The request data for actions such as: signing up and logging in, modifying their account details (i.e. their password), deleting or modifying students and evaluators accounts, deleting or editing details of courses (assigning instructors to a course), dropping students from courses; modifying instructor access to course management abilities by changing the role of an instructor to a teaching assistant or vice versa is sent to the respective processes to trigger them, and once the data processing is completed by the respective process, the data generated is sent to the data store where it will be stored for all processes to use. There is once instance within these key processes for an administrator user, where the data store will have been populated with the required data for a process which is the process of restoring deleted student, evaluator and course data and being able to view it instantly post-restore. This process will be a data retrieval processes from the data store, indicating the flow of data out from the data store through a process where processing is completed, and the data is visible to the appropriate external entity via the User Interface.

System Features

Our project is comprised of the following major features (features with an asterisk next to them indicate that I have contributed to the development of that feature):

*Feature 1: An easily navigable and user-friendly login and signup webpage for all user roles (student, instructor, and administrator) (Fully functional with backend, no bugs)

- The login and signup webpages allow for recovering passwords with a forgot password feature (to be elaborated in the next section) having validation on the frontend and backend for what kind of passwords are accepted and are resistant against SQL injection attacks where the attacker tries to type in SQL statements to try and query sensitive data within our database onto the frontend.
- The user interface is made so that it is more accessible to people who are color-blind, and we have implemented a hybrid color of having a color contrast on the sides for people who can see it, and white towards the middle for color blind people to be drawn to the login and signup sections of the respective webpages.
- The login and signup webpages feature allows for routing based on a reverse proxy (Nginx) established for conditional rendering of webpages based on the user role (detailed in the System Architecture section above).

Feature 2: An easily navigable and user-friendly edit account details webpage for users of all roles (student, instructor, and administrator) to modify their account's details (Fully functional with backend, no bugs)

- Users of all roles are able to click on the hamburger menu on the right hand top corner of the webpage once they are logged in and access the Account tab, which they can click to be redirected to the Account details page.
- For all users, they can change their passwords only from here, the rest of the information (name and email) cannot be changed from here.
- Only the administrator has exclusive access to change the name, and email of a user if any situations warrant the need for a change of name, and the email of student or instructor.

*Feature 3: Easily navigable and user-friendly webpages to join courses from as a student, and to create courses for instructors (Fully functional with backend, no bugs)

- Student users will be able to see a list of all created courses by any instructor and will be able to search up which course they would like to join.
- Students can also see the course names and course codes of each course that is available to be joined on top of course cards that are clickable.
- When a student clicks on a course card, they get a confirmation popup on whether they would like to join the selected course card's respective course, and they can confirm or deny it, if they confirm it, they will be enrolled in the course of the course card that they clicked on successfully with a green confirmation toast.

*Feature 4: Comprehensive Session management and storage that is compatible with our frontend (Fully functional with backend, no bugs)

- All frontend webpages will be pulling session variables set in the backend using Express.js for each session a user initiates when they log in to our application.
- The backend session variables will be stored for the session duration until they log out of the application.
- All frontend functionalities will use session storage for a user to access data on their frontend pages that is linked to their user ID.

*Feature 5: Easily navigable and user-friendly webpages for all users who forgot their password to use to reset their password (Fully functional with backend, no bugs)

- This feature includes a forgot password page where they can get an email sent to reset their password, and a reset password page where the user can change their password with their email.
- The user receives an email in their inbox on clicking reset password after typing in their email twice to confirm.
- The email sent will contain a link that they can paste into their browser and be able to reset the password for the account associated to their email, after which, the user is taken to the login page.

*Feature 6: An easily navigable and user-friendly webpage component to perform automated grading of submissions with the AI API (Fully functional with backend, no bugs)

- This feature contains a button that when clicked on by an instructor or teaching assistant can start the automatic grading of all the submission files or links submitted by students with the help of OpenAI's Assistants API.
- The AI is fed an answer key (if provided by the instructor), and a rubric for an assignment for which submissions need to be graded, as well as an AI prompt (if provided by the instructor) that can establish subjective or objective measures for grading submissions to match the grading style of the instructor.
- The response of the AI is directly displayed on the instructor's end as an AI-generated text response and integer grade.

*Feature 7: Easily navigable and user-friendly webpages to Create and edit an Assignment's details (Fully functional with backend, no bugs)

- This feature includes a frontend page that allows instructors (not teaching assistants) to create assignments with a name for the assignment, a rubric (mandatory), an assignment key (optional), a due date, the number of marks the assignment is out of, and the ability to reuse a past rubric.
- When the assignment details are filled out, the user can click on the post button to make the assignment available to students.
- This feature also includes the publish assignment page that gives instructors the ability to change the due date, reupload a new answer key, and change the assignment name and the rubric.

*Feature 8: User-friendly and automated notification emails for a new assignment's creation and an assignment being graded (Nearly fully functional, 1 small bug)

- This feature has been architected to automatically send an email to all students within a course when a new assignment has been created in a course.
- This feature also includes automatically sending an email to a student when their assignment submissions have been graded and released by the instructor.
- The emails contain details of the assignment created or the course and assignment when a student's submission has been graded depending on the notification email sent (the small bug associated with this feature is mentioned below in detail in the Features Individually completed section)

Feature 9: Easily navigable and user-friendly webpage containing all the assignment-wise grade averages and class grade average (Fully functional with backend, no bugs)

- This feature includes a frontend page that dynamically updates and displays the assignment-wise grade averages as and when more assignments get created and graded.
- At the top right corner of this page, the class grade average is also rendered for the instructor to get a good assessment of students' performance on assignments throughout the course.
- This page has a table containing the name of the assignment, the highest grade obtainable on that assignment, the due date, along with the current average grade on that assignment.

*Feature 10: An easily navigable and user-friendly webpage containing all the necessary features to edit the AI-generated grades and feedback (Fully functional with backend, no bugs)

- This feature includes a frontend page that displays all the AI-generated grades and feedback, with a box beneath the AI feedback for the instructor to input their feedback using the AI's feedback as a guide.
- The instructors can use markdown in their feedback to give the student an exact idea of where they lost marks for their submissions and make it more readable on the student's end.
- This feature also gives instructors and teaching assistants the ability to type in a final grade based on the AI's grade for a submission, and they can click the 'mark assignment as complete' button at the bottom of the page to save the grades for the student along with the feedback.

Feature 11: Easily navigable and user-friendly webpage components to release or unrelease assignments (Fully functional with backend, no bugs)

- This feature includes allowing instructors and teaching assistants the ability to release an assignment to students or unrelease it for any reason.
- Releasing and unreleasing an assignment will populate or remove the assignment on a student's grades page as well as their assignments page.
- The grades associated with an assignment that has been unreleased are excluded from the class average calculation on the instructor's grades page.

*Feature 12: Easily navigable and user-friendly webpages for students to submit their submission files or links for an assignment and download their submitted files if needed (Fully functional with backend, no bugs)

- This feature includes a webpage where students can access the instructions (rubric) for an assignment, the number of marks it is out of, and boxes where they can click to upload a link or submission files in accepted formats.
- This page where the students can submit assignments will also be where they can see their feedback after the instructor has released the final grades with feedback and will be the page they will be redirected to when the click on an Assignment's name on their grades page.
- This feature also includes a submissions page for the student where students can download their uploaded submission files if they would like to see what the instructor and the AI will see when they will be marking the student's submissions or double check if they uploaded the correct files.

*Feature 13: Easily backend-compatible and non-cloud solution for file storage to perform CRU (Create, Read, and Update) operations on assignment submission files and assignment key files (Fully functional with backend, no bugs)

- This feature includes setting up a file storage system that stores uploaded files from students for assignment submissions, and assignment keys from instructors.
- This feature allows for the upload of a limited number of files from a student and an instructor and performing file validation on the frontend before uploading files to file storage.
- Students can reupload submissions if the file name of a newly uploaded file is the same as one that has been previously uploaded and allows for reuploading of an answer key file by an instructor for an assignment to allow for error recovery for both students and instructors.

Feature 14: Easily navigable and user-friendly user account management webpages from an administrator's view (Fully functional with backend, no bugs)

- This feature includes being able to edit both student and instructor accounts, and edit details like email, and name.
- This feature allows administrators to also switch the account type of an account from a professor to a teaching assistant and vice versa.
- This feature also allows for restoring deleted student and instructor accounts.

Feature 15: Easily navigable and user-friendly administrator view webpage to register a new professor or teaching assistant (Fully functional with backend, no bugs)

- This feature allows administrators to manually create new accounts for instructor users and appends the new account to the list of instructors, from where the account details can be managed by the administrator.
- The administrator can set the type of instructor account to be created, of which there are two types: instructor (with exclusive access to all instructor features like course management and being able to create courses), and teaching assistants (restricted access to course management and course creation features).
- This feature also shows a confirmation that the user was registered and redirects to the page with the list of instructors, from where an administrator can search for an instructor by name.

Features Individually Completed

I worked on a total of 10 out of the 15 major features briefly described above. My feature-wise contributions are as follows (I have mentioned larger Pull Requests from our project's repository linked to the work that I contributed to these features for convenience):

Feature 1: An easily navigable and user-friendly login and signup page for all user roles (Fully functional, no bugs) (Pull Requests #158 & #329)

- For this feature, I worked on implementing the frontend validation for user input fields on the login and signup pages for the student view, as well as adding in a security measure on the frontend that protects our database from SQL injection attacks on login and signup pages for all user roles.
- The details of what I completed when going about implementing the frontend validation of user input are as follows:
    - Created a regex pattern that checks for different symbols in the input fields for the email and makes sure that the user enters a valid email first for the student login page along with empty string checks rendering errors for empty fields in the email and password field for the student login page.
    - Added in password creation validation by only allowing passwords that were 6 characters or longer and must contain only alphabets and digits using another regex pattern that checks and matches the entered password for the requirements after which it displays validation errors when signing up as a student.
- The details of what I completed when going about implementing the SQL injection attack protection are as follows:
    - Created a regex pattern that covers majority of the possible keywords within general SQL syntax.
    - Added string checks for both email and password fields on all login and signup pages (for all user roles) to ensure that if a user types in any of the SQL keywords defined in the regex pattern and clicks the login or signup button, then the typed in text is matched against the regex pattern and the user sees an invalid input error thrown on the frontend.

Feature 3: Easily navigable and user-friendly webpages to join courses from as a student, and to create courses for instructors (Fully functional, no bugs) (Pull Request #468)

- This feature was partially broken, so I worked on fixing it. Before I fixed it, an instructor user had to click the create course button 6 times before the course got created and toast popup confirmed the course creation. This error was happening due to a duplicate course Id attempting to be inserted into the database, more specifically, course Ids 1 to 5 already existed because of the dummy data that had been inserted into the database for testing purposes.
- The details of what I completed when going about implementing the fix for this 6-time click create course error are as follows:
    - Added code towards the end of our data definition language file (init.sql) where the I set the values of all relations with serial primary keys (auto incrementing primary key) to start one value after the current highest id of that relation, therefore, avoiding insert conflicts when creating a new course. What this essentially did was shift the position where insert statements would begin inserting data to the database exactly after the dummy data's serial course Ids. Our dummy data currently has 5 courses in total and was not working before because those insert operations for a new course were attempted on all those already existing course Ids, and the feature would begin to work on the 6^{th} course Id insert.
    - Added in a function to set the course Id of the newly created course in the backend from an existing axios post request containing the course code and course name from the input fields on the frontend to a backend route controlling the creation of courses. The response from the backend route contained the course Id itself, allowing for it to be set in session storage for the current session and be accessible. This was done as a measure to ensure that the course Id was being set cleanly and would not cause further errors on top of the serial sequence shifting mentioned above.

Feature 4: Comprehensive Session management and storage that is compatible with our frontend (Fully functional, no bugs) (Pull Request #468)

- For this feature, I worked on adding in functions that set the session data for pages whose operations required data that was specific to that session such as the instructor ID and course ID, to different files such as the create assignment page, course home page, publish assignment page, and the evaluator's view of the list of students in a course page.
- The details of what I completed when going about implementing the setting and ensuring of session data for pages whose operations required data that was specific to that session such as the instructor ID and course ID are as follows:
    - Added in a function to set the course Id when required to fetch the list of students in a course using an axios post request containing the course Id obtained from the URL parameters and set in the backend's session storage in a call to a backend route that took care of the functionality of fetching the list of students in a course. This was then set in session storage, which could be used from the frontend for the setting session data and changing it if you switched a course to a different one to see the list of students in another course. I took a similar approach to setting the variables in session storage where they can be used in different functionality for different pages such as the publish assignment page, where the focus was to set both the instructor and course IDs, and similarly on the create assignment page as well. For the course home page, it was a similar approach to setting the list of students as I had to find a way to set the course Id in session storage. Adding these functions in for these files helped remove 400 status code errors for bad requests.
    - Added in a function that logged the set session data on the console and was visible using Inspect element to confirm that the session data being set was correctly to pinpoint the causes of any errors when fixing backend routes for these pages.

Feature 5: An easily navigable and user-friendly webpage for all users who forgot their password to use to reset their password (Fully functional, no bugs) (Pull Request #171)

- For this feature, I worked on writing the backend functionality for the forgot password webpage from a student's view along with the reset password backend and frontend from a student's view, which users could access by clicking the forgot password text on the login pages for all the user roles (forgot password page), and entering their correct email associated with their account and hitting send reset email. They would then receive an email with a link they can open, from where they can reset their password and login to access the Aivaluate platform successfully.
- The details of what I completed when going about implementing the backend functionality for the forgot password and reset password, as well as writing the frontend for the reset password page are as follows:
    - Created a notification email sent using the Brevo email API from a Gmail address made specifically for our project (aivaluateofficial@gmail.com). Once a valid email is entered for an existing user in our database from the forgot password page, a reset password link is sent to the user with a random string of length 20 as part of the reset password URL. An update is done in the database to the reset password temporary token that is set based on this random string, and the expiration time for the link is 1 hour, giving the user ample time to click on the link and reset their password. They can copy the link sent in the email available in their inbox (or in the spam folder), and paste it into their browser, where they will be allowed to reset their password.
    - Added in frontend validation for the reset password frontend page to disallow passwords that are not of length of 6 characters or greater, and do not have a letter and a number within the password. The rest of the frontend validation was done similar to how Feature 1 was implemented to ensure that a semi-complex password was set, and they would have to re-enter their password to confirm it. The email of the user is checked against the database to see if an account with that email exists and throws an error if an account with the email entered does not exist, so that the users know they can sign up to access our application.

Feature 6: An easily navigable and user-friendly webpage component to perform automated grading of submissions with the AI API (Fully functional, no bugs) (Pull Requests #468 & #479)

- For this feature, I worked on writing the backend functionality that generated the AI response and integrated with the existing selected assignments frontend page containing the 'Grade with AI' button for instructors to use. The instructor or teaching assistant could click on an assignment name that they want to grade and that

would redirect them to the page that displays all the submissions from all students for that particular assignment (selected assignments frontend page) and can automatically grade all submissions by clicking the button mentioned above which uses the AI model I configured to work with our backend evaluator service to grade submissions. A toast popup will show up once the AI finishes grading the submissions after clicking the Grade with AI button.

- The details of what I completed when going about implementing the backend functionality for the communication with the AI API to complete automatic grading are as follows:
    - Created a frontend call to a backend route (/process-submissions on backend-eval routes folder's assignmentRoutes.js file) that passed over session variables like assignment ID, course ID, and instructor ID to the backend-ai container (where AI grading will take place) by sending an axios post request to the backend-ai container on port 9000 from the above-mentioned backend-eval route (to /process-submissions on backend-ai folder's aiRoutes.js file). This AI route communicates back and forth with the axios request and response from the AI once the grading is successfully complete.
    - Created 2 functions: 1 function that processes an individual student's entire set of assignment submission files and links together by calling a helper function (processStudentSubmissions) before moving to the next student's submissions after providing the instructor ID, course ID, and assignment ID as parameters to the function (called processSubmissions) which also sets up the creation of an assistant object for the AI API communication as well as passing the assistant the rubric, answer key, max points, and instructor-selected prompt are all obtained through SQL queries, and the submissions for one student. There is a for loop that runs over all student Ids within the course to grade all submissions of each student. The second function (processStudentSubmissions) is called by the processSubmissions function once for each student and takes in the following as parameters: the student id, the submission files for the student, the max points, the rubric, the answer key, and the prompt provided by the instructor. The processStudentSubmissions function is where the AI receives the submission files and the assignment key as OpenAI file objects which can be easily parsed by the AI API. The function then checks if the submission was a link or a file, if it was a file, then the files are uploaded directly to the API as OpenAI file objects, if it is a GitHub gist link (static website, so the DOM is easier to extract and grade compared to dynamic websites like Google Docs), then the DOM of the webpage is extracted as text from the body tag in the HTML of the GitHub Gist link and inserted into a text file that is sent to OpenAI as a file object. Two Boolean values are set to true initially to check the status of whether the links and submissions failed to be graded after all retries. I have implemented a retry mechanism that after 10 retries, defaults to sending a null grade and the default feedback to the database. The retry mechanism is triggered when the AI response is not parsed correctly. When communicating with the AI API in text format as a message, the rubric, answer key, the prompt, and maximum points available on the assignment are all sent to the AI as part of a text message to let the AI know which files are submissions and which one is the answer key. To initiate the AI communication, I created an OpenAI thread, attached the text message to the AI onto the thread, created an OpenAI run instance, attached the run to the thread, retrieved the result of the run to check the progress of the AI grading, retrieved the run, popped the stack of the list of messages with the AI to get the most recent AI response (I prompted the AI API to respond with the grades and feedback in a JSON format for easier parsing) and sent the message to the response parsing function called parseAIResponse which does the cleanup of the response and parsing of the AI response. It tries with plaintext matching first, and if the plaintext matching fails, regex matching and JSON parsing exist as a backups, and if the parsing fails with all approaches, then the default feedback and grade are assigned and sent to the database to notify the instructor that manual grading is required. If the parsing is successful, the AI grade and feedback is extracted from the JSON response and sent to the database to be inserted in the appropriate relation (AssignmentSubmission) holding the AI feedback text and AI grades for all submissions for one student. This feedback and grade generated by the AI can then be pulled to the frontend for the grading assignment page, where overriding or using and editing the AI generated grades and feedback is done as well as marking and releasing final instructor grades with feedback. Once all the submissions are graded, the response is communicated back to the process-submissions route in the same aiRoutes.js file within the backend-ai container, which passes the response back to the /process-submissions backend route in the backend-eval container and is sent to the frontend to be displayed as a toast message.

Feature 7: Easily navigable and user-friendly webpages to Create and edit an Assignment's details (Fully functional, no bugs) (Pull Request #468)

- For this feature, I worked on writing the backend and the frontend that controlled an instructor's ability to be able to create an assignment with uploads and reuploads of answer keys enabled for all instructors. The instructor can upload or reupload a single answer key file in any of the following file formats: .html, .css, or .js. The reupload feature allows for quick error correction in the case that the instructor uploaded the wrong answer key.
- The details of what I completed when going about implementing the backend & frontend functionality for the answer key uploads and reuploads are as follows:
    - Added backend logic that used multer's disk storage to store the temporary file paths and use them to create unique file paths for where the assignment keys would be stored in the docker volumes. The assignment keys needed to be stored in docker volumes as our client instructed us to not use a cloud-based file storage solution. I created new file paths for every answer key file based on their course ID, instructor ID, and assignment ID, and stored the files in the docker volumes at this file path: app/aivaluate/backend-eval/assignmentKeys/courseId/instructorId/assignmentId.
    - Added frontend session management for instructor ID and course ID, allowing for files to be uploaded as form data appended to an axios post request sending all the assignment details to the backend route at /assignments/assignmentId/solutions (located in the assignmentRoutes.js file in the backend-eval directory) which controls the insertion of file paths of the answer keys within the docker volumes. I also added the same frontend session management for the assignment key reuploads on the publish assignment frontend page, and once again allowed for the reuploaded files to be uploaded as form data appended to an axios post request sending all assignment details to the backend route at /assignments/assignmentId (located in the assignmentRoutes.js file in the backend-eval directory) which updates the file path of the assignment key to the newly uploaded file's file path from the docker volumes.

Feature 8: User-friendly and automated notification emails for a new assignment's creation and an assignment being graded (Fully functional, 1 bug) (Pull Requests #500 & #503)

- For this feature, I worked on writing the backend and database code for automated notifications to be sent out to all students enrolled in a course when a new assignment has been created and when the submissions for an assignment have been graded for a student and are visible on their end with the final feedback from the instructor. Emails are sent out from our application's main email address (aivaluateofficial@gmail.com).
- The details of what I completed when going about implementing the backend and database code for sending automatic notification emails of created assignments and graded assignments are as follows:
    - Added 2 functions and triggers at the end of the data definition language (DDL) file (init.sql). The triggers are activated when a new assignment gets created and when the grades for an individual student are released by an instructor by clicking 'mark as complete' for all assignment submissions belonging to a student after assigning a final grade and feedback.
    - Activated the first trigger on new entries into the database for the Assignment relation. I have configured a listener function in the EvalServer.js file which listens for the message 'assignment_created' through the PostgreSQL database connection pool established. The 'assignment_created' message is defined in the notify_assignment_creation function within the DDL file. This message also contains the course Id along with the assignmentId of the course to which the newly inserted assignment belongs in, and this is sent along with the 'assignment_created' message as a JSON object. This JSON object is parsed by the listener function (listenForAssignmentCreation) and sent to a function (sendAssignmentCreationEmails) that sends the email. The parsed courseId along with the assignmentId are used in intermediate SQL queries and joins needed to be done to obtain the list of students' emails who need to be notified. This function calls the sendMail function that uses the Brevo email API configured above in the EvalServer.js file to send out the emails successfully.
    - Activated the second trigger by an update to any of the rows in the AssignmentGrade relation in the database. I have configured a listener function in the EvalServer.js file which listens for the message 'assignment_graded' through the PostgreSQL database connection pool established. The

'assignment_graded' message is defined in the notify_student_after_grading function within the DDL file. This message also contains the email of the student, the name of the course, name of the assignment, along with the courseId where the assignment grade of a student was released, and this is sent along with the 'assignment_graded' message as a JSON object. This JSON object is parsed by the listener function (listenForAssignmentGrading) and sent to a function (sendAssignmentGradedEmail) that sends the email. The parsed email of the student, the name of the course, name of the assignment, and the courseId where the assignment grade of a student was released are used directly in notifying the list of students that their assignment has been graded. This function calls the sendMail function which uses the Brevo email API configured above in the EvalServer.js file to send out the emails successfully. The small bug with this feature is that it sends out emails prematurely sometimes when a submission is graded for one student, all of them get notified, and it may not have been graded yet for the other students who got the notification email.

Feature 10: Easily navigable and user-friendly webpage containing all the necessary features to edit the AI-generated grades and feedback (Fully functional, no bugs) (Pull Request #468)

- For this feature, I worked on writing the backend code that allowed for final grades and feedback to be captured from the frontend grading assignments page where the final grades and feedback from the instructor are entered. This is done with an axios put request routed to the backend route at /assignment/complete/:studentId/:assignmentId, where if a grade doesn't exist there, then it will insert the instructor grade and feedback for the most recent submission based on an assignmentSubmissionId, assignmentId, and studentId. This is done for all submission files and links graded by the instructors and teaching assistants. A for loop run on the submissions ensures the same grade for all submissions linked to a student for an assignment. This is viewable for all submissions from the instructor's view on the Grading Assignments frontend page by pulling from the backend using an axios get request. This get request displays the assignment submission details such as the AI feedback and the AI score. This feature also allows for multiple regrading with the AI, and multiple updates on the AI feedback, grades, final feedback and final grades in the database. I also implemented the feature where the individually submitted file name and extension (or link) that appears on the File directory on the Selected Assignments frontend page is what is clickable taking the user to the grading assignments page for that submission with that specific file being downloadable, or if it is a link, then a redirect takes you to the GitHub Gist page if clicked on.

Feature 12: Easily navigable and user-friendly webpages for students to submit their submission files or links for an assignment and download their submitted files if needed (Fully functional, no bugs) (Pull Requests #479 & #317)

- For this feature, I worked on writing the backend code for allowing students to submit 7 files in the accepted formats mentioned above, added empty file checks, and used multer's disk storage to map the assignment submissions to the docker volume mounted onto the students backend container. This was done by simply using multer's disk storage as done in Feature 7 to construct temporary file paths for the assignment submissions based on student ID, course ID, and assignment ID. The submissions were then stored in the assignmentSubmissions docker volume, and these file paths were recorded to refer to from the database in the AssignmentSubmission relation for each file uploaded (reuploading a file with the same name and extension replaced an uploaded file with the same name and extension). I wrote the frontend code for this feature to send the uploaded files as an array of file paths via an axios post call to the student backend.

Feature 13: Easily backend-compatible and non-cloud solution for file storage to perform CRU (Create, Read, and Update) operations on assignment submission files and assignment key files (Fully functional, no bugs) (Pull Request #317)

- For this feature, I worked on writing the backend code and docker compose file code for setting up our file storage solution that was not cloud-based, requiring persistent storage of assignment submissions and answer keys for viewing as well as AI processing. This was done by storing and using temporary file paths (stored in database) established with multer's disk storage that mapped to where the files were stored on the docker volumes. After mounting the volumes for the submission files and answer keys onto the appropriate backend containers where they are pulled from, the files at the specified file paths were accessible and updateable.

Installation and Setup

The following instructions can be followed to run and test out our project:

1. Install the Visual Studio Code Integrated Development Environment to run the application's code locally.
2. Install Docker Desktop for your device to run the application as a containerized application.
3. Install GitHub Desktop for your device to be able to easily locate and clone the source code of our application onto your device.
4. Go to our team's GitHub repository link: https://github.com/UBCO-COSC499-Summer-2024/team-8-capstone-team-8, and click on the green button that says 'Code' in the upper-middle part of the page to the right, and you will see a link and a button to the right of the link on the dropdown that says Copy URL to clipboard, click the button to copy the link to our repository.
5. Navigate to the directory where you have installed GitHub Desktop, Docker Desktop, and Visual Studio Code.
6. Click on the installed Docker Desktop application on your device, perform whatever necessary setup is required by your operating system, and give it a few seconds to get initialized and start the Docker engine.
7. Once the Docker engine has been started, click on the installed Visual Studio Code application on your device, and give it a few seconds to start up, and there is no need to install any extensions or follow any other guide that pops up when you first open Visual Studio Code.
8. Once you have opened Visual Studio Code, click on the installed GitHub Desktop application on your device, and click Clone a Repository, and then select the third option to the screen's center right that says URL, and paste the URL that you copied earlier here by pressing Command + v for Apple or Ctrl + v for Windows.
9. This popup will give you the option to choose where to store the project's source code on your device so please choose a destination folder that is easily accessible to you.
10. Visit the OpenAI website and sign up. Once logged in, go to the APIs section and pay 5$ or the minimum amount of money to use the API.
11. Generate a secret API key with OpenAI and copy the API key to your clipboard with Command + v for Apple or Ctrl + v for Windows.
12. Go back to Visual Studio Code and click on File on the top left corner of your screen, and select open folder, and find the directory where the cloned repository code is and click to open that folder in Visual Studio Code.
13. Now with Visual Studio Code, simply hold down the following keys at once to open a Command Palette:
    a. Command + Shift + p if you are using an Apple device or,
    b. Ctrl + Shift + p if you are using a Windows device
14. Type the text "terminal", and the first option that shows up on the dropdown should say "View: Toggle Terminal".
15. Click on the first option in the dropdown, and it will open a terminal on the bottom of the screen for you to type commands and run code from, which should be automatically set to the directory where your project is located. If not, use the cd (change directory) command to get to the target directory where the cloned project source code has been downloaded to.
16. Open the file titled docker-compose.yml within the app directory of the project source code and hold down Command + f for an Apple device, or Ctrl + f for a Windows device. Type in the text ".env" and you should see 4 matches with the directories where the environment variable files are located, locate those files and find the text that says 'OPENAI_API_KEY=' in all of them, and paste your OpenAI secret API key next to this text into the .env files located in both the backend-ai and backend-eval directories under the app/aivaluate directory (do not insert quotes around the secret API key).
17. Type in the text "docker-compose up --build" or "docker compose up --build" into your terminal to begin running the application and wait for the application's containers to build fully and you should see 4 messages that say "Server is running on PORT" where PORT will be 4 separate numbers 9000, 3000, 6000, and 4000 (for the different backend components of our application), and the frontend should build without breaking. The application has now been successfully built and you can see the green containers on Docker to confirm.
18. Now, type in http://localhost:5173/stu/login if you would like to login/signup as a student, http://localhost:5173/eval/login if you would like to login as an instructor, and http://localhost:5173/admin/login if you would like to sign up or login as an admin.
19. The init.sql file in the database folder of the source code contains test credentials you can use to login, the general credentials we have set for now is testprof/testta/admin/chinmay@email.com, password: pass123 (use anything between slashes followed by @email.com to login and access our web application).

Lessons Learned and Project Reflections

My personal experience of working on this project over the past 3 months is that it has taught me many important lessons in the software development process, challenges that arise during a project and amicably proposing solutions to problems, my personal and team growth, and in terms of individual lessons learned. In terms of the software development process, I have learned to frequently create commits of my code to the remote repository as soon as even parts of code are working to maintain a version of your branched code that is working in case you need to erase changes or run into issues and need to restart. I have also learned to service Pull Requests thoroughly explaining the feature that was worked on and what level of testing it has undergone with my code that was developed using Test-Driven Development, and how to test the feature before reviewers can review it. Another lesson I learned and wished we as a team had done earlier in the project was strictly following the Test-Driven Development approach. In terms of planning of issues for the project board, we did run into some team problems with some members not contributing equally like the rest of the team, and we needed to adjust for them not doing work by us doing their work and having to put in more hours to keep development going at the expected pace to keep our burnup chart on the right track. In terms of implementation, our team had only one member who had taken the COSC 360 (Web Programming) course here at UBC, and most of us needed to learn majority of the technology stack including the nuances of building a web application that containerized multiple services, causing our software to take longer to be of expected quality for a capstone project, but we kept putting in effort throughout the semester to improve our product as much as we could. Iteratively developing our project also caused us some issues with our team members not merging pull requests early causing pile up of functionality, and hurried merging would break features that previously worked. We did learn from this issue, and towards the latter half of the project most of us began to review pull requests quicker and put in relevant comments where required after testing a branch. Our team dynamic was a little lopsided and became our biggest challenge as the term progressed, by the quarter way point of the summer term, 3 team members including myself had to complete majority of the work of the project because the other 2 members of the group did not show initiative to make this project better than what we currently have nor did they seem to have the desire to make this project a success. The major contributors of the team gained familiarity with the technology stack being used for our project towards the latter half of the project, removing the initial block we had in our development cycles (first 4-5 weeks) leading to our cycle reports being all over the place. Our product meets majority of the initially set requirements and expectations. In our requirements checklist meeting with the Professor, we missed out on lesser weighted requirements like giving instructors the option to choose what type of links are accepted as assignment submissions links from students. The possible reason for that could be that I was putting a lot of hours into the project and was very burnt out, so we could have delegated this feature to a different team member. The caveat for our team was only 2 other members were putting in more effort than myself and they were busy with their own features and couldn't help me cover the requirements we ended up missing. I was not confident the other 2 team members could get the feature completed and neither did they volunteer to pick up a feature as difficult as that. So, I picked it up myself and gave my best effort. Over the course of this project, I have been able to advance my knowledge in JavaScript, Express.js, Node.js, React.js, web development in general, as well as learn new skills in developing large-scale projects such as containerizing services in our system and learning how to build a microservices-based architecture with inter-container communication using requests and data to make our system interactive and visible to the user via the UI. The development experience that I have gained from this project, as well as the lessons I learned (mentioned above) will help me build confidence in developing architecture-based scalable software in teams. It also lets me know that there is so much more to learn in software development the longer I spend thinking about it in team settings. Another lesson I learned during the course of this project was that my expectations from the 2 team members who did not show as much initiative could have caused them extra pressure to deliver on their features, leading to them giving up on their work at times. I acknowledge that I can find better ways to communicate with my team members what is expected of them by coming to a middle ground on the work that is expected from them, and slowly but incrementally give them bigger features to complete to build confidence in them, to continue working towards our common goal of making the project a success.

References

1. Team 8 Project Plan (https://github.com/UBCO-COSC499-Summer-2024/team-8-capstone-team-8/blob/master/project-plan.md)
2. Team 8 Design Plan (https://github.com/UBCO-COSC499-Summer-2024/team-8-capstone-team-8/blob/master/design-plan.md)
3. Team 8 GitHub Repository (https://github.com/UBCO-COSC499-Summer-2024/team-8-capstone-team-8)