Bilkent University

Department of Computer Engineering

CS 491

Senior Design Project

CODED.

2023-24 Fall Semester

Analysis and Requirements Report

Project ID: T2322

Project Team Members:

Zeynep Hanife Akgül	(22003356)
Beyza Çağlar	(22003394)
Selin Bahar Gündoğar	(22001514)
Emre Karataş	(22001641)
Zeynep Selcen Öztunç	(21902941)

Project Advisor: Halil Altay Güvenir

Instructors: Atakan Erdem, Mert Bıçakçı

Innovation Expert: Bora Güngören

08 December 2023

Table of Contents

1. Introduction

Bilkent University is currently facing a problem of finding lab tutors for the CS (Computer Engineering) and CTIS (Information Systems and Technologies) labs. Most students have questions about the lab assignments or have trouble debugging a specific part of their code during their lab periods. Thus, these students ask their questions to tutors and teacher assistants (TAs). Unfortunately, only 2-3 TAs and 1-2 tutors can be present during each lab session. The CS courses do not have trouble finding TAs for lab sessions as there is a graduate program in the CS department; though, the CTIS department has to rely on successful students to volunteer to fill the tutor vacancies for TA roles as the CTIS department does not have a graduate program. Furthermore, tutors for both CS and CTIS courses are selected among volunteering students who display proficiency in the course's coding language, and most tutors cannot stay throughout all of the lab hours due to their schedules. Even when they are able to do so, the student-to-tutor ratio is around 1/30 [1]. Henceforth, tutors do not have the time to pay attention to each student. Furthermore, hundreds of non-STEM (Science, Technology, Engineering, and Mathematics) students take mandatory coding courses offered by the CS department every semester, and non-STEM students are far more likely to ask trivial questions about the lab compared to STEM students. Therefore, there is an urgent lab tutor needed in the CS and CTIS departments. Nevertheless, this crisis can be solved with our application called "*CODED.*".

CODED. is a website application that assists students, tutors, TAs, and instructors in CS and CTIS labs throughout the lab process. The application answers student questions about the lab assignments through its chatbot property. It leads students in the coding process by guiding them to write clean code and checking for any code smells, which is "any characteristic in the source code of a program that possibly indicates a deeper problem."[2].

Followingly, the completed lab assignments are then uploaded to the application, where these assignments are tested for plagiarism based on the criteria that the instructor selects. The instructor can compare the code to a specific section, the whole class, code on Github, past codes, or check for AI (Artificial Intelligence) generated code similarity. Furthermore, the code will be passed by test cases provided by the instructor or created by the chatbot. The tool will grade the code based on the given criteria in a fair manner by looking at the main logic of the code and running every method separately if needed to check for correctness. Furthermore, the code will be statically checked for partial point calculations. The instructors will be given an insight into how the grading was tested and which sections of the code led to errors to take off points. The students will be able to view their grading report if the instructors allow it on the application, and students will be able to refute their generated grades. Therefore, the application will tremendously shorten the time it takes to grade the code and lighten the workload of tutors, TAs, and teaching assistants.

2. Current System

There is no identical application to CODED.;though, we were inspired by similar existing apps when making design choices for our application. Foremost, the most similar application on the market is Khan Academy's chatbot tutor application called Khanmigo. Khanmigo is mostly used for math tutoring; nevertheless, it also guides students with their questions about coding too. However, the application does not include a platform for universities to upload and manage lab assignments and, therefore, cannot be considered as an existing "current system" for us. Moreover, this application is still in its beta version and is not developed enough to have a wide impact on the tutor chatbot market for labs and coding assignments. The second application we have considered similar to ours is ChatGPT, which is a chatbot that can answer any question about the desired topic. Its ability to answer coding

related questions and write code is also widely acknowledged and used. Nevertheless, ChatGPT answers the questions directly whereas our application aims to answer student questions in a way that does not directly give the answer but rather shows the way for the student to learn about and complete the assignment. The third application we were inspired by is Moodle as it is a course management system, a system that we have also included in our application. Though, our application's main focus is improving the student lab experience by helping them write code. Additionally, we have noted from ChatGPT that there is a legal requirement to include a privacy policy if the chatbot is collecting data from its users [3]. We will not be directly collecting user data from the chatbot; though, we will still be guided by the GDPR as we still keep user data from code analytics and personal information such as email etc.

3. Proposed System

3.1. Overview

CODED. is a comprehensive web application designed to address the challenges faced by Bilkent University in providing sufficient lab tutoring for CS and CTIS courses. The platform aims to improve the lab experience by facilitating efficient student-tutor interactions and streamlining the code evaluation process. Key features include a chatbot for immediate answers to student questions, automated code quality checks, plagiarism detection, and an advanced grading tool. The application not only helps students refine their coding skills but also significantly reduces the workload for tutors and TAs by automating critical aspects of the lab process, such as testing for code correctness and calculating grades and partial points. Moreover, the integrated chatbot significantly enhances the learning process by providing instant, personalized support for students' questions, making it easier to deal with coding

challenges. Using advanced technologies, CODED. establishes a new level of effectiveness for educational tools, offering a more efficient and improved learning experience for Bilkent University's CS and CTIS departments.

3.1.1. Test Case Runner

The Test Case Runner component is essential for evaluating student-submitted code. It allows for the grading of lab assignments using custom test cases provided by instructors or teaching assistants. The tool analyzes the core logic of the code and tests each method individually for accuracy. It also conducts static checks for partial points. Instructors receive detailed insights into the grading process, including error identification, while students can access their grades and refute them if permitted by instructors. The system can also generate these test cases on demand, ensuring a comprehensive assessment of each student's work.

3.1.2. Chatbot

The chatbot is integral for student interaction, providing a platform where students can ask questions and get guidance on coding problems. It supports file reading, which allows students to submit their code directly for detailed assistance. The chatbot is carefully designed to facilitate learning and problem-solving, guiding students without giving away direct answers, thus upholding academic integrity.

3.1.3. Code Quality Check

The Code Quality Checker component analyzes submissions for potential issues and overall quality and its main focus is focused on improving the quality of student code. It accepts code through both file uploads and direct text input. By highlighting areas for improvement, the component aids students in adopting coding best practices and refining their code before final submission.

3.1.4. Plagiarism Check

The Plagiarism Checker component checks the originality of student submissions. It compares each submission against others and against a pre-existing code database and generates a similarity rate. Based on this similarity rate, the code is suggested to be plagiarized or not. This plays a key role in upholding academic standards, as it helps identify similarities that may suggest plagiarism, thereby ensuring the uniqueness and integrity of student work.

3.2. Functional Requirements

- The user must register/login via their email address and password.
- Only instructors can register for the application.
- The system will utilize 2FA authentication during registration, where an email will be sent to the email address provided by the user.
- The user can log out of the system.
- The instructor will be uploading an Excel file to register the students to the course.
- Tutors, graders, and TA's will be registered to the application by the instructor when the instructor is creating a course.
- The students can ask questions to the chatbot as long as they are logged in.
- The students can test their codes in the lab with a restricted reach to a limited number of test cases. However, they cannot see the contents of the test cases, only the results.
- The students must submit their codes during the hours specified by the instructor or TA for evaluation.
- The students can use the test case generator and chatbot outside of lab hours for practice. (Optional)
- The students can view their past submissions and grades.

- The instructors can upload new lab assignments and specify lab hours.
- The instructors can create new classes and sections for classes.
- The instructors and TAs can assign test cases for the final evaluation of the lab or for students to test their codes.
- The instructors and TAs can adjust the test cases generated by the system.
- The chatbot can read files, meaning that the students can send their code as a file to the chatbot to ask questions to the chatbot.
- The code analyzer can accept files as well as copy-pasted text.
- The system can generate test cases on demand.
- The chatbot cannot directly give the solution to the problem asked by the student.
- The system can grade the student's code based on customized test cases.
- The system gives partial grading to student's code using unit tests.
- The system must check the codes uploaded by students for similarities between other students or codes in the database.
- The system must not allow students to use the test case generator during lab hours.
- After running test cases for the instructors and TAs, the system will give feedback.
- The student will get a plagiarism report after the deadline is over.
- The student will be able to object to the grading.

3.3. Non-functional Requirements

3.3.1. Usability

The website will have a consistent user interface to ensure a clear navigation and to make it effortless for users to have a plain understanding of the fundamentals of the program. There will always be a navigation bar on every page so that reaching any main feature will require only one operation. Other than reaching the main tools, any feature of the system will be accessed within at most three clicks or operations. Chatbot, which is one of the most

significant features, will be accessible and easily seen on any page as it will be on a slider view. This focus on user-centric design aims to make the website user-friendly and engaging.

3.3.2. Reliability

The system will be available to all users at all times, including lab hours and practicing hours, with minimal downtime. More specifically, the website will maintain an availability of at least 99.8% over a year, with a total downtime of no more than 9 hours annually. High-security databases will be used to prevent data loss and ensure the safe storage of personal data. The website will provide clear error messages to inform users in cases of problems. Errors will be logged for analysis and debugging purposes.

3.3.3. Performance

The system will be responsive even during most busy periods like lab hours. It will be able to handle the usage of multiple users without a significant performance slowdown. It is aimed that the response time of the system will be under 0.5 seconds for any standard user action and transactions and will be under 2 seconds for more complex tasks, even when the website experiences a peak usage time. Regarding the fact that a standard lab has at most 100 students, the system will be able to maintain its performance with at least 200 users using the application at the same time. There are several APIs that will be used during the development process, so it is aimed that the system will ensure that the third-party API calls have a latency of less than 1 second.

3.3.4. Supportibilty

The website will be a modular system to ensure an easy integration of new features. This design will allow changes to be made without disrupting the entire system such that the integration of at least 10 new features per year will be supported without causing any significant corruption to the existing system. The system will be maintained as long as it is being used, and the maximum response time for user support inquiries related to new and

existing features will be 24 hours. Furthermore, the website will support at least 3 different web browsers and any device that is able to use web browsers to surf through websites.

3.3.5. Scalability

We are targeting a large audience using the system at the same time since students participate in the lab during the same hours. Thus, the significance of the scalability increases so that the application will easily adapt and will be able to expand in response to growing demands. As the user volume increases, the performance and responsiveness of the system will remain consistent, and for this purpose, we set a threshold that the website will accommodate at least 50% growing demand in terms of the number of users and content volume. The importance of scalability will increase in the future as the system is developed to allow for usage from different schools or even companies, increasing the volume of content of the application.

3.3.6. Security

The system will not store much personal information but only the names, Bilkent school IDs, and email addresses. While it is crucial to store this information safely, the system should also be secure to store the work students upload and their respective grades. To ensure the security of this data, we will employ encrypted databases, utilizing industry-standard encryption algorithms to safeguard sensitive information. The system will also implement several actions to strengthen security. User access to the system will be strictly controlled, limiting access to authorized personnel only: enrolled students, instructors, teaching assistants, and tutors. Furthermore, in the cases of security breaches, the system will have an incident response time of no more than 30 minutes.

3.4. Pseudo Requirements

- GitHub and Jira will be used to track the deadlines, issues, and code.

- React.JS framework will be used in the frontend.
- Spring Boot framework and Python (for chatbot development) will be used for the backend.
- PostgreSQL hosted by Amazon RDS server will be used for the database system.
- GitHub Pages will be used to host the application.
- SonarQube will be used to make clean code and code convention checks.
- CODE-LLAMA-7B model with a fine-tuned version developed by the *CODED*. team will be used for the chatbot.
- MOSS will be used in checking the codes uploaded by students for plagiarism.
- pytest will be used to run test cases on Python codes.
- UnityTest or CUnit will be used to run test cases on C codes. However, as there are many options in the market, other APIs or frameworks may be used due to compatibility issues.

3.5. System Models

3.5.1. Scenarios

Scenario 1: Instructor Registration

- **Actors:** Instructors
- **Entry Conditions:** Instructors open the app and select "Register."
- **Exit Conditions:** Instructors either complete registration OR exit with a failure.
- **Flow of Events:**
    - Instructors fill in the registration form.
    - Instructors submit a form and receive a verification email.
    - Instructors verify email and complete registration OR instructors need to complete registration form again.

Scenario 2: User Login

- **Actors:** Registered users, 2FA Service
- **Entry Conditions:** User opens the app and clicks the login button.
- **Exit Conditions:** User logs in OR asks the enter credentials again.
- **Flow of Events:**
    - User enters an email and password.
    - User clicks "Login." button.
    - 2FA Service authenticates if conditions are satisfied and the system redirects to the Dashboard page. Otherwise, the system asks the credentials again.

Scenario 3: Password Reset

- **Actors:** Registered users, Email Service
- **Entry Conditions:** Registered user selects "Forgot Password.".
- **Exit Conditions:** Password is reset OR process is exited with a failure.
- **Flow of Events:**
    - Registered user enters registered email.
    - Email Service sends an encrypted reset link mail, specified for that specific user.
    - User sets a new password via the link OR the system generates error and directs to the error page.

Scenario 4: Course Creation

- **Actors:** Instructors
- **Entry Conditions:** Instructor decides to create a course and clicks the “Create Course” button.

- **Exit Conditions:** Course is created or process is exited with a failure.
- **Flow of Events:**
    - Instructor enters course details.
    - Instructor submits the course.
    - System confirms course creation OR generates errors and directs to the error page.

Scenario 5: Assignment Submission

- **Actors:** Students
- **Entry Conditions:** Student selects an assignment to submit.
- **Exit Conditions:** Assignment is submitted or process is exited with a failure.
- **Flow of Events:**
    - Students upload the assignment file.
    - Students click the "Submit" button.
    - System asks “Are you sure?” question in pop-up modal.
    - System confirms submission OR generates error and directs to the error page.

Scenario 6: Assignment Evaluation

- **Actors:** Instructors, TAs, Test Case Runner, Code LLama Model, Email Service
- **Entry Conditions:** Assignments submitted by students are ready for evaluation.
- **Exit Conditions:** Assignment evaluation is completed.
- **Flow of Events:**
    - Actor reviews the submissions.
    - Actors assign the grades, by also considering late submission and/or plagiarism situations to deduct base grade if necessary.

- Students' grades are updated automatically. In the case of plagiarism, necessary actors are notified with Email Service to take possible actions.

Scenario 7: Chatbot Assistance

- **Actors:** Students, Code LLama Model
- **Entry Conditions:** Students ask their corresponding questions to the chatbot.
- **Exit Conditions:** Chatbot answers the questions OR chatbot generates error messages.
- **Flow of Events:**
  - Student writes his/her question to the input chat box OR the student can choose pre-generated questions to ask to the chatbot.
  - Chatbot thinks about the question and answers according to it OR generates an error message if such error happens.
  - Students are allowed to ask further questions to the chatbot, and if that is the case, the chatbot answers these questions accordingly.
  - Students might evaluate the answer and if needed may report the chatbot's answer if they are not appropriate or correct.

3.5.2. Use Case Model

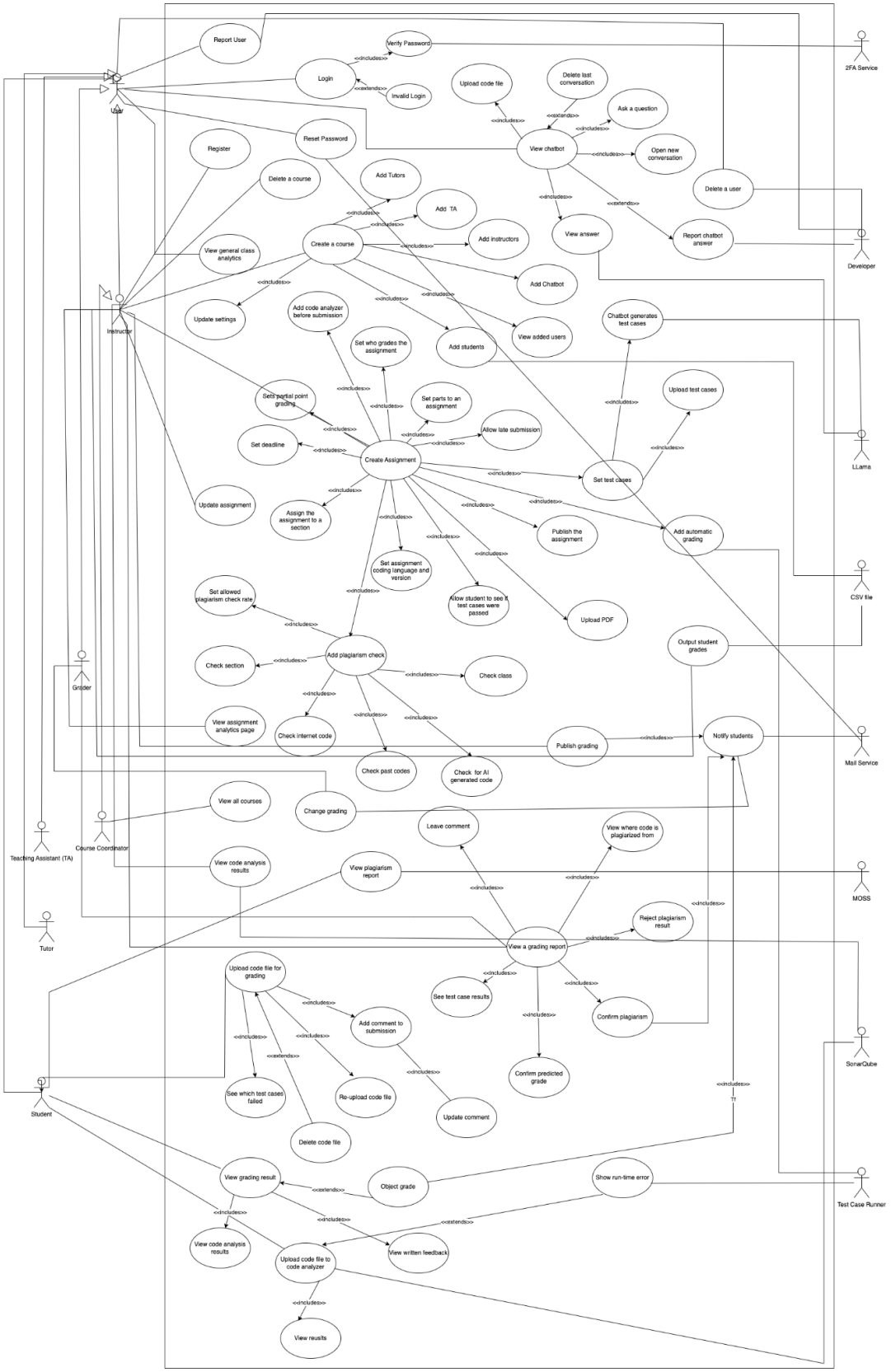

Fig. 3.5.2.1: Use Case Model

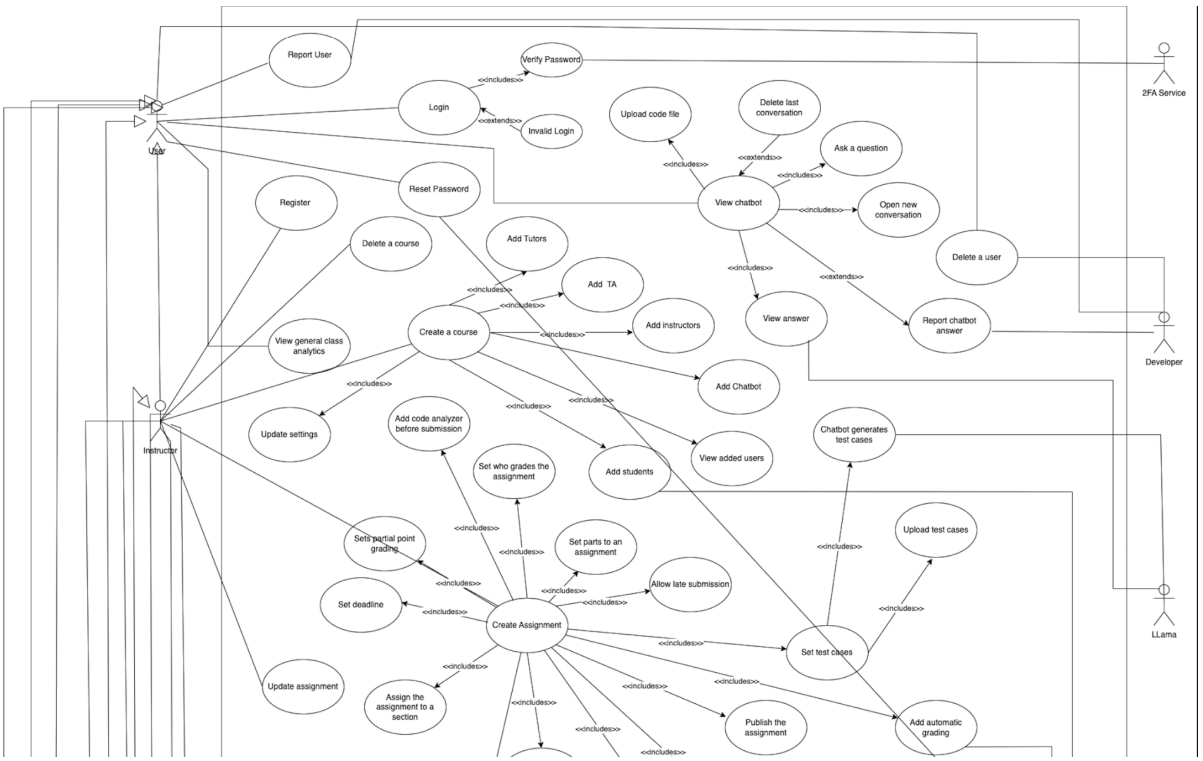

Fig. 3.5.2.2: Use Case Model Close up 1

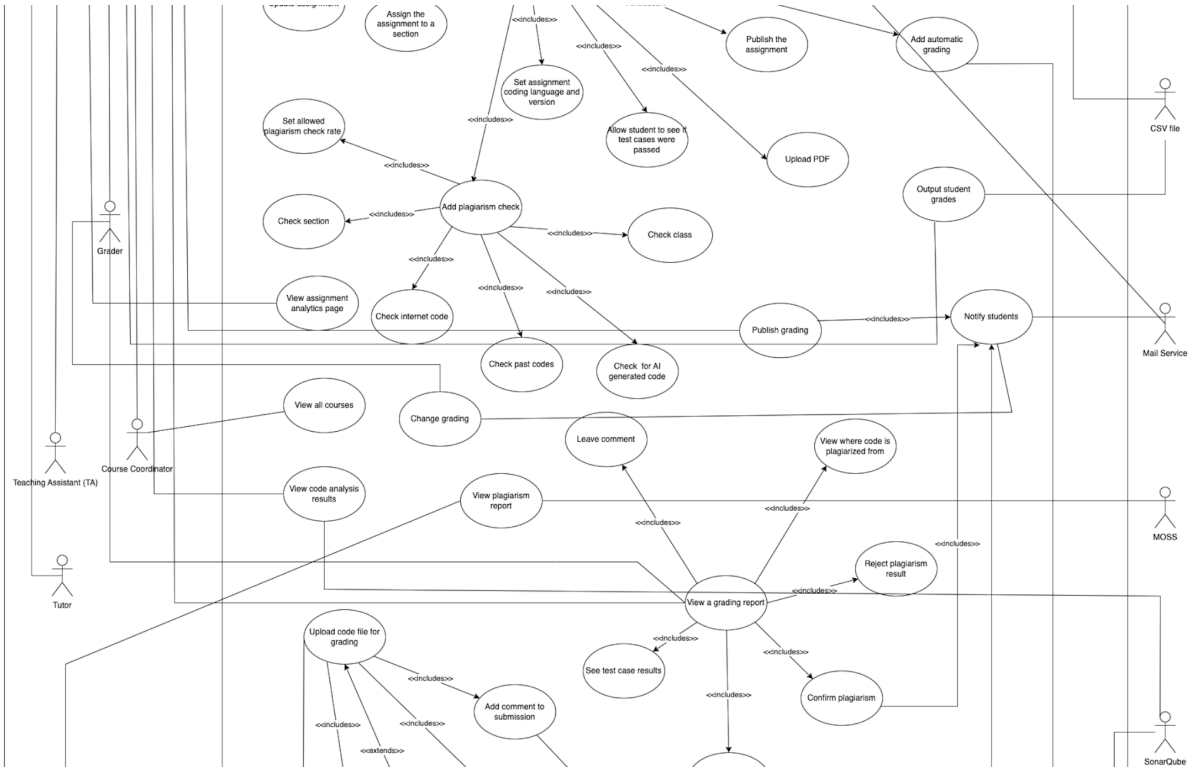

Fig. 3.5.2.3: Use Case Model Close up 2

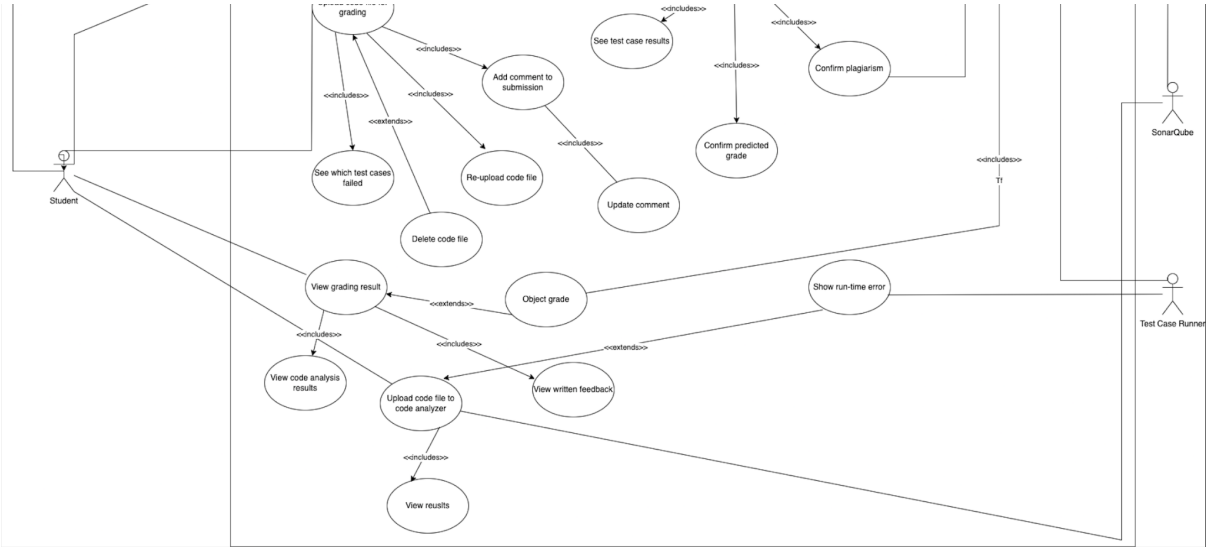

Fig. 3.5.2.4: Use Case Model Close up 3

The link of the use case diagram:

https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=use_case.drawio#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1qLfFazorBzIvEquQP1iQLHa2Qa8mdD0Z%26export%3Ddownload

Use Case: Login

Participating Actor(s): User

Entry Conditions: The actor is not logged in.

Exit Conditions: The actor provides email and password.

The Flow of Events:

- Actor clicks on email/password fields.
- Actor provides the necessary information.
- If information is valid, the actor logs in *CODED.* application.

Use Case: Verify Password

Participating Actor(s): User, Email Service

Entry Conditions: The user provides his/her email and password.

Exit Conditions: The user verifies his/her password correctly.

The Flow of Events:

- User gets a verification message from the Email Service.
- User provides a verification code to the *CODED*.
- If the code is valid, the user logs into *CODED*.

Use Case: Invalid Login

Participating Actor(s): User

Entry Conditions: The actor provides the wrong password or ID.

Exit Conditions: This use case extends the Login use case. It is initiated by the system whenever the User provides the wrong ID and/or password to the system.

The Flow of Events:

- Actor tries to log in.
- Actor enters invalid information into the required fields.
- The system generates an error message, and the actor must re-enter the faulty information.

Use Case: Reset Password

Participating Actor(s): User

Entry Conditions: The user wants to reset the password.

Exit Conditions: The user resets his/her password successfully.

The Flow of Events:

- User clicks on the reset password button and enters email.
- User gets a reset password email from the E-mail service.
- User enters the code onto the reset page.
- User enters the new password.
- User's password has been changed.

Use Case: Report User

Participating Actor(s): User

Entry Conditions: A user reports another user.

Exit Conditions: The reported user's information is sent to the developer/application system.

The Flow of Events:

- User clicks the report user button.
- User explains the reasons behind reporting.
- Report details are sent to the developer.

Use Case: Register

Participating Actor(s): Instructor

Entry Conditions: An instructor clicks on the register button.

Exit Conditions: The instructor registration information is sent to the application.

The Flow of Events:

- Instructor clicks on register button
- Instructor enters personal information.
- Instructor information is saved and the account is created.

Use Case: View Chatbot

Participating Actor(s): User

Entry Conditions: User clicks on the chatbot button.

Exit Conditions: User clicks on another view page button.

The Flow of Events:

- User clicks on the chatbot button

Use Case: Upload Code File

Participating Actor(s): User

Entry Conditions: User is on the chatbot page and clicks on the upload code button.

Exit Conditions: User question is answered based on the uploaded file.

The Flow of Events:

- User clicks on the upload code button.
- User selects the code file to be uploaded (must be a file with code extension).
- User enters the question about the code and presses enter.
- Chatbot analyzes the file and answers the question accordingly.

Use Case: Upload Code File

Participating Actor(s): User

Entry Conditions: User is on the chatbot page and clicks on the upload code button.

Exit Conditions: User question is answered based on the uploaded file.

The Flow of Events:

- User clicks on the upload code button.
- User selects the code file to be uploaded (must be a file with code extension).
- User enters the question about the code and presses enter.
- Chatbot analyzes the file and answers the question accordingly.

Use Case: Delete last conversation

Participating Actor(s): User

Entry Conditions: User is on the chatbot page and clicks on the delete conversation button.

Exit Conditions: Chat is deleted.

The Flow of Events:

- User clicks on the delete conversation button on any previous chats listed on the left side of the screen.
- System asks if the user wants to delete it for sure.
- User clicks on proceed button
- The previous chat becomes deleted.

Use Case: Ask a question

Participating Actor(s): User

Entry Conditions: User is on the chatbot page and writes a prompt.

Exit Conditions: The prompt is answered.

The Flow of Events:

- User writes the questions and presses enter.
- Chatbot answers the question appropriately.

Use Case: Open a new conversation

Participating Actor(s): User

Entry Conditions: User is on the chatbot page and writes a prompt.

Exit Conditions: The prompt is answered.

The Flow of Events:

- User writes the questions and presses enter.
- Chatbot answers the question appropriately.

Use Case: Delete a user

Participating Actor(s): User, Developer

Entry Conditions: User clicks on delete account.

Exit Conditions: User is deleted.

The Flow of Events:

- User clicks on the delete user button.
- The user is deleted by the system/developer.

Use Case: Report chatbot answer

Participating Actor(s): User, Developer

Entry Conditions: User clicks on report answer button.

Exit Conditions: Report is sent to the developer/system.

The Flow of Events:

- User reports an answer sent by the chatbot.
- The report is received by the developer/system.

Use Case: View answer

Participating Actor(s): User, Llama

Entry Conditions: User sends a prompt to the chatbot.

Exit Conditions: Answer is sent by Llama to the chatbot.

The Flow of Events:

- User asks a question to the chatbot.
- Chatbot takes the question to the Llama server.
- Llama replies back to the chatbot.

Use Case: Create a course

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create a course button.

Exit Conditions: Course is created.

The Flow of Events:

- Instructor clicks on the create a course button.
- Instructor enters the mandatory information about the course and completes the process.
- System creates the course.

Use Case: Add tutors

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create a course button. This use case has an includes relationship with **Create a course** use case.

Exit Conditions: Tutors are added.

The Flow of Events:

- Instructor adds information about the tutors.
- System creates accounts for those tutors if they do not already exist in the system.
- System adds the tutors to the created course.

Use Case: Add TA

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create a course button. This use case has an includes relationship with **Create a course** use case.

Exit Conditions: TA's are added.

The Flow of Events:

- Instructor adds information about the TA's.
- System creates accounts for those TA's if they do not already exist in the system.
- System adds the TA's to the created course.

Use Case: Add Instructor

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create a course button. This use case has an includes relationship with **Create a course** use case.

Exit Conditions: Other instructors are added.

The Flow of Events:

- Instructor adds information about the instructors.
- System creates accounts for those instructors if they do not already exist in the system.
- System adds the instructors to the created course.

Use Case: Add chatbot

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create a course button. This use case has an includes relationship with **Create a course** use case.

Exit Conditions: Chatbot and its properties are set.

The Flow of Events:

- Instructor sets the chatbot properties.

Use Case: Add students

Participating Actor(s): Instructor, CSV File

Entry Conditions: Instructor clicks on the create a course button. This use case has an includes relationship with **Create a course** use case.

Exit Conditions: Students are added to the created course.

The Flow of Events:

- Instructor uploads a CSV file which contains a formatted list of students that will be registered to the course.
- New accounts will be created for those students whose emails are not present in the application.
- The students will be added to the course.

Use Case: Update settings

Participating Actor(s): Instructor

Entry Conditions: Instructor has created a course before.

Exit Conditions: Instructor updates the new settings.

The Flow of Events:

- Instructor inputs the new course settings.
- New settings are saved on the application.

Use Case: View added students

Participating Actor(s): Instructor

Entry Conditions: Instructor has uploaded the CSV file.

Exit Conditions: Instructor views the list of added users.

The Flow of Events:

- Instructor uploads the CSV file which contains student information.
- The student information is extracted from the file and the student list is given to the instructor on the application.

Use Case: View general class analytics

Participating Actor(s): User

Entry Conditions: User is registered to a course and an assignment has been completed.

Exit Conditions: User views the class analytics.

The Flow of Events:

- User clicks on the view class analytics page.
- User can view the class analytics for each completed assignment.

Use Case: Delete a course

Participating Actor(s): Instructor

Entry Conditions: Instructor is assigned to the course as the course coordinator.

Exit Conditions: Instructor deletes the course.

The Flow of Events:

- Instructor presses on delete course button.
- Are you sure popup comes up.
- Instructor deletes the course permanently.

Use Case: Create an assignment

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create assignment button.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor clicks on the create assignment button.
- Instructor inputs the assignment information.
- Assignment is created.

Use Case: Set deadline

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor clicks on the set deadline property.
- Instructor sets the deadline property by picking a date and time for it.

Use Case: Set partial point grading

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor clicks on the partial point grading property.
- Instructor sets if partial points are allowed in the grading of the assignment.

Use Case: Set who grades the assignment

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor clicks on the grader properties.
- Instructor sets the grader people as the graders.

Use Case: Add code analyzer before submission

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor chooses to add or eliminate code analyzer tools that students can use before their submissions.

Use Case: Set parts to an assignment

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor adds parts to an assignment and sets a grade percentage for each part.

Use Case: Allow late submission

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor allows late submission to be made and can add a grading penalty.

Use Case: Assign the assignment to a section

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor selects which section/s the assignment will be assigned to.

Use Case: Set assignment coding language and version

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor selects the coding language and the version of the code for the specific assignment.

Use Case: Allow students to see if test cases were passed

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor sets the test case property to allow the students to view the test cases.

Use Case: Upload PDF

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor uploads the assignment PDF to the assignment.

Use Case: Set test cases

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor chooses the specific test cases to grade the assignment.

Use Case: Chatbot generates test cases

Participating Actor(s): Instructor

Entry Conditions: Instructor is setting the test cases.

Exit Conditions: Instructor has set the test cases.

The Flow of Events:

- The instructor chooses the chatbot to generate the test cases and sees which test cases are created by the chatbot.
-

Use Case: Upload test cases

Participating Actor(s): Instructor

Entry Conditions: Instructor is setting the test cases.

Exit Conditions: Instructor has set the test cases.

The Flow of Events:

- The instructor uploads the test case code.

Use Case: Add automatic grading

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor adds the automatic grading property to the assignment.

Use Case: Add plagiarism check

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- Instructor sets the uploaded codes to be checked for plagiarism.

Use Case: Set allowed plagiarism check rate

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the add plagiarism button. This use case has an includes relationship with the Add **plagiarism check** use case.

Exit Conditions: Instructor has set plagiarism properties.

The Flow of Events:

- Instructor sets a rate that shows which uploaded student codes will be considered plagiarized.

Use Case: Check section

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the add plagiarism button. This use case has an includes relationship with the Add **plagiarism check** use case.

Exit Conditions: Instructor has set plagiarism properties.

The Flow of Events:

- Instructor makes plagiarism checks between sections.

Use Case: Check past codes

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the add plagiarism button. This use case has an includes relationship with the Add **plagiarism check** use case.

Exit Conditions: Instructor has set plagiarism properties.

The Flow of Events:

- Instructor makes plagiarism checks between past classes' codes and current uploads.

Use Case: Check internet code

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the add plagiarism button. This use case has an includes relationship with the Add **plagiarism check** use case.

Exit Conditions: Instructor has set plagiarism properties.

The Flow of Events:

- Instructor sets the plagiarism settings so that students' codes are compared to codes found on the internet.

Use Case: Check class codes

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the add plagiarism button. This use case has an includes relationship with the Add **plagiarism check** use case.

Exit Conditions: Instructor has set plagiarism properties.

The Flow of Events:

- Instructor sets the settings so that there is plagiarism checks between classmates' codes.

Use Case: Check for AI generated code

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the add plagiarism button. This use case has an includes relationship with the Add **plagiarism check** use case.

Exit Conditions: Instructor has set plagiarism properties.

The Flow of Events:

- Instructor sets the settings so that the student codes are checked for AI generation.

Use Case: Publish assignment

Participating Actor(s): Instructor

Entry Conditions: Instructor clicks on the create an assignment button. This use case has an includes relationship with **Create an assignment** use case.

Exit Conditions: Instructor has created an assignment.

The Flow of Events:

- The instructor sets the publishing time of the assignment and publishes the assignment.

Use Case: Update assignment

Participating Actor(s): Instructor

Entry Conditions: Instructor has created an assignment before.

Exit Conditions: Instructor settings have been updated.

The Flow of Events:

- The instructor updates the properties and sets the new settings by clicking on the save button.

Use Case: Publish grading

Participating Actor(s): Instructor

Entry Conditions: Uploaded codes are graded and approved by graders.

Exit Conditions: Grade notification is sent by email.

The Flow of Events:

- The instructor clicks on the publish grade button.
- Emails are sent to students to let them know grading results.

Use Case: Notify students

Participating Actor(s): Instructor, Mail Service

Entry Conditions: An event has happened and students must be notified by email.

Exit Conditions: Emails are sent to students.

The Flow of Events:

- The instructor creates a trigger for emails to be sent.
- Students receive an email indicating the event.

Use Case: Output student grades

Participating Actor(s): Instructor, CSV File

Entry Conditions: Student grades are announced.

Exit Conditions: Student information and grades are outputted as a csv file.

The Flow of Events:

- The instructor clicks on the output student grades button.
- The student information and grades are put in a csv file and downloaded to the computer.

Use Case: View assignment analytics page

Participating Actor(s): Instructor

Entry Conditions: That specific assignment's deadline is passed.

Exit Conditions: Assignment analytics are shown on the website.

The Flow of Events:

- The instructor clicks on the assignment analytics page.
- Analytics are shown on the page.

Use Case: View all courses

Participating Actor(s): Course Coordinator

Entry Conditions: There are multiple sections to a course or the course coordinator is an instructor of another course.

Exit Conditions: All courses list is shown.

The Flow of Events:

- The instructor clicks on the view all courses page and views the list.

Use Case: Change grade

Participating Actor(s): Grader

Entry Conditions: An assignment must have already been graded.

Exit Conditions: The grade is changed.

The Flow of Events:

- The grader goes onto the grading page and clicks on the change grade button.
- The grader enters the new grade and presses on the update grade button.
- The student is notified with an email on the update.

Use Case: View plagiarism report

Participating Actor(s): User, Moss

Entry Conditions: An assignment must have already been graded and the plagiarism check was completed.

Exit Conditions: The plagiarism report is viewed.

The Flow of Events:

- The user views the plagiarism report created on a student's assignment upload. Students can only view their own plagiarism reports.

Use Case: View code analysis results

Participating Actor(s): Instructor, SonarQube

Entry Conditions: The student must have already uploaded the assignment solution.

Exit Conditions: The code analysis is shown on the page.

The Flow of Events:

- The instructor clicks on the individual student's code analysis result button.
- The code analysis results are taken from the Sonarqube API and displayed on the page.

Use Case: Upload code file for grading

Participating Actor(s): Student

Entry Conditions: The assignment must be published.

Exit Conditions: The student code is successfully uploaded.

The Flow of Events:

- The student clicks on the upload assignment button.
- Student picks a file out with a code extension and uploads it.
- The upload has been completed.

Use Case: See which test cases failed

Participating Actor(s): Student

Entry Conditions: The code assignment must be uploaded by the student to proceed with this use case. It has an includes relationship with the **Upload code file for grading** includes.

Exit Conditions: The student views the failed test case results.

The Flow of Events:

- The student clicks on view failed test cases button.
- THe failed test cases are listed on the page.

Use Case: Delete code file

Participating Actor(s): Student

Entry Conditions: The code assignment must be uploaded by the student to proceed with this use case. It has an extends relationship with the **Upload code file for grading** includes.

Exit Conditions: The uploaded code file is successfully deleted.

The Flow of Events:

- The student clicks on the delete submission button.

- The popup asks if the student wants to delete the submission.
- Student accepts and the submission is deleted.

Use Case: Re-upload code file

Participating Actor(s): Student

Entry Conditions: The code assignment must be uploaded by the student to proceed with this use case. It has an includes relationship with the **Upload code file for grading** includes.

Exit Conditions: The uploaded code file is successfully updated.

The Flow of Events:

- The student clicks on the update submission button.
- Student chooses a new file to upload and the student presses on the proceed button.
- The popup asks if the student wants to update the submission.
- Student accepts and the submission is updated.

Use Case: Add comment to submission

Participating Actor(s): Student

Entry Conditions: The code assignment must be uploaded by the student to proceed with this use case. It has an includes relationship with the **Upload code file for grading** includes.

Exit Conditions: The comment is added to the submission.

The Flow of Events:

- The student clicks on the add comment button.
- Student writes the comment and presses the save button.

Use Case: Update comment

Participating Actor(s): Student

Entry Conditions: The student must have added a comment to the submission to proceed with this use case. This use case has an includes relationship with the use case **Add comment to submission**.

Exit Conditions: The comment is added to the submission.

The Flow of Events:

- The student clicks on the update comment button.
- Student edits the comment and clicks on the save button.

Use Case: View a grading report

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded to view this page.

Exit Conditions: The grading report is viewed.

The Flow of Events:

- The application automatically does the grade approximation, plagiarism check, and check for test cases.
- Grader and the instructor can see this report when they click on the view grading report.

Use Case: Leave comment

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded to view this page. This use case has an includes relationship with the use case **View a grading report.**

Exit Conditions: The comment is added to the report.

The Flow of Events:

- The grader/instructor clicks on the leave a comment button.
- The comment is written and the grader/instructor presses on save button.
- Comment is added to the grading report.

Use Case: View where code is plagiarized from

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded to view this page.

This use case has an includes relationship with the use case **View a grading report.**

Exit Conditions: The source of plagiarism is shown to the grader/instructor.

The Flow of Events:

- The grader/instructor clicks on the see plagiarism origin button.
- The integrated MOSS interface is shown on the page.

Use Case: See the test case result

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded to view this page.

This use case has an includes relationship with the use case **View a grading report.**

Exit Conditions: The test case result is shown on the screen.

The Flow of Events:

- The grader/instructor clicks on the see test case result button.
- The test case results are shown on the screen.

Use Case: Confirm predicted grade

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded to view this page. This use case has an includes relationship with the use case **View a grading report.**

Exit Conditions: The student grade is finalized.

The Flow of Events:

- The application makes a predicted score for the assignment based ont he test cases and static code analysis.
- The grader/instructor may choose to accept or change this grade.
- The grader/instructor accepts or inputs a new grade and presses the save button.

Use Case: Confirm plagiarism

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded and has to fail the plagiarism test. This use case has an includes relationship with the use case **View a grading report.**

Exit Conditions: The plagiarism status is finalized and the student is notified.

The Flow of Events:

- The application makes a predicted plagiarism rate.
- Grader/instructor confirms the case as plagiarism.
- An email is sent to the student to notify the student.

Use Case: Reject plagiarism

Participating Actor(s): Grader, Instructor

Entry Conditions: The assignment has to be automatically graded and has to fail the plagiarism test. This use case has an includes relationship with the use case **View a grading report.**

Exit Conditions: The plagiarism status is rejected and the student is notified.

The Flow of Events:

- The application makes a predicted plagiarism rate.
- Grader/instructor rejects the plagiarism result.

Use Case: View grading result

Participating Actor(s): Student

Entry Conditions: The assignment grading is published.

Exit Conditions: The student views the grading result.

The Flow of Events:

- The student clicks on the grading result page.

Use Case: Object grade

Participating Actor(s): Student

Entry Conditions: The assignment grading is published. This use case has an includes relationship with the use case **View grading result**.

Exit Conditions: The student objects to the assigned grade.

The Flow of Events:

- The student clicks on the object grade button and enters the reason for the objection.
- The objection request is sent to the grader/instructor as an email to notify them.

Use Case: View code analysis results

Participating Actor(s): Student

Entry Conditions: The assignment grading is published. This use case has an includes relationship with the use case **View grading result**.

Exit Conditions: The student views the code analysis results.

The Flow of Events:

- The student clicks on the view code analysis result page.
- The code analysis results are shown on the page.

Use Case: View written feedback

Participating Actor(s): Student

Entry Conditions: The assignment grading is published. This use case has an includes relationship with the use case **View grading result**.

Exit Conditions: The student views the written feedback sent by the grader/instructor.

The Flow of Events:

- The student clicks on the view code analysis result page and scrolls down the page.
- The written feedback is displayed, which was sent as a comment by the grader/instructor.

Use Case: Upload code file to code analyzer

Participating Actor(s): Student, SonarQube

Entry Conditions: The code analyzer has to be allowed by the instructor to use it.

Exit Conditions: The student has successfully uploaded the code file and analyses results are received.

The Flow of Events:

- The student clicks on the code analyzer page and then clicks on the upload file button.
- A code extension file is uploaded to the application.
- The code is run on SonarQube and the results are sent back to the application to be displayed.

Use Case: View results

Participating Actor(s): Student

Entry Conditions: The code analyzer has to be allowed by the instructor to use it. This use case has an included relationship with the **Upload code file to code analyzer** use case.

Exit Conditions: The student views the results.

The Flow of Events:

- The student clicks on the code analyzer page and then clicks on the view results.
- The results are shown on the page.

3.5.3. Object and Class Model

The class diagram given below displays the general structure of CODED. There are 18 classes in total.

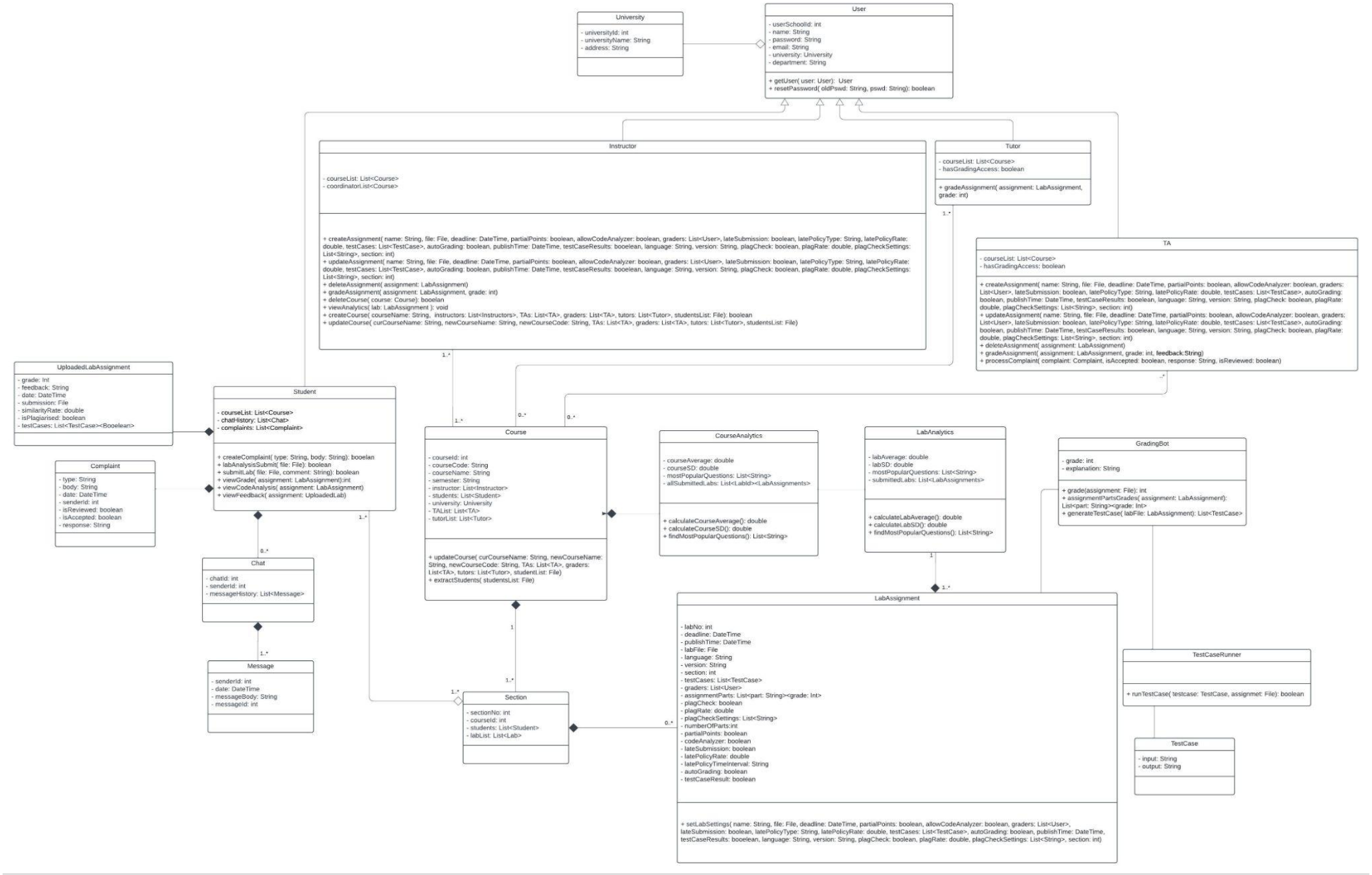
University
- universityId: int
- universityName: String
- address: String
User
- userSchoolId: int
- name: String
- password: String
- email: String
- university: University
- department: String
+ getUser(user: User): User
+ resetPassword(oldPswd: String, pswd: String): boolean
Instructor
- courseList: List<Course>
- coordinatorList<Course>
+ createAssignment(name: String, file: File, deadline: DateTime, partialPoints: boolean, allowCodeAnalyzer: boolean, graders: List<User>, lateSubmission: boolean, latePolicyType: String, latePolicyRate: double, testCases: List<TestCase>, autoGrading: boolean, publishTime: DateTime, testCaseResults: booelean, language: String, version: String, plagCheck: boolean, plagRate: double, plagCheckSettings: List<String>, section: int)
+ updateAssignment(name: String, file: File, deadline: DateTime, partialPoints: boolean, allowCodeAnalyzer: boolean, graders: List<User>, lateSubmission: boolean, latePolicyType: String, latePolicyRate: double, testCases: List<TestCase>, autoGrading: boolean, publishTime: DateTime, testCaseResults: booelean, language: String, version: String, plagCheck: boolean, plagRate: double, plagCheckSettings: List<String>, section: int)
+ deleteAssignment(assignment: LabAssignment)
+ gradeAssignment(assignment: LabAssignment, grade: int)
+ deleteCourse(course: Course): boolean
+ viewAnalytics(lab: LabAssignment): void
+ createCourse(courseName: String, instructors: List<Instructors>, TAs: List<TA>, graders: List<TA>, tutors: List<Tutor>, studentsList: File): boolean
+ updateCourse(curCourseName: String, newCourseName: String, newCourseCode: String, TAs: List<TA>, graders: List<TA>, tutors: List<Tutor>, studentsList: File)
Tutor
- courseList: List<Course>
- hasGradingAccess: boolean
+ gradeAssignment(assignment: LabAssignment, grade: int)
TA
- courseList: List<Course>
- hasGradingAccess: boolean
+ createAssignment(name: String, file: File, deadline: DateTime, partialPoints: boolean, allowCodeAnalyzer: boolean, graders: List<User>, lateSubmission: boolean, latePolicyType: String, latePolicyRate: double, testCases: List<TestCase>, autoGrading: boolean, publishTime: DateTime, testCaseResults: booelean, language: String, version: String, plagCheck: boolean, plagRate: double, plagCheckSettings: List<String>, section: int)
+ updateAssignment(name: String, file: File, deadline: DateTime, partialPoints: boolean, allowCodeAnalyzer: boolean, graders: List<User>, lateSubmission: boolean, latePolicyType: String, latePolicyRate: double, testCases: List<TestCase>, autoGrading: boolean, publishTime: DateTime, testCaseResults: booelean, language: String, version: String, plagCheck: boolean, plagRate: double, plagCheckSettings: List<String>, section: int)
+ deleteAssignment(assignment: LabAssignment)
+ gradeAssignment(assignment: LabAssignment, grade: int, feedback:String)
+ processComplaint(complaint: Complaint, isAccepted: boolean, response: String, isReviewed: boolean)
UploadedLabAssignment
- grade: Int
- feedback: String
- date: DateTime
- submission: File
- similarityRate: double
- isPlagiarised: boolean
- testCases: List<TestCase><Booelean>
Student
- courseList: List<Course>
- chatHistory: List<Chat>
- complaints: List<Complaint>
+ createComplaint(type: String, body: String): booelan
+ labAnalysisSubmit(file: File): boolean
+ submitLab(file: File, comment: String): boolean
+ viewGrade(assignment: LabAssignment):int
+ viewCodeAnalysis(assignment: LabAssignment)
+ viewFeedback(assignment: UploadedLab)
Complaint
- type: String
- body: String
- date: DateTime
- senderId: int
- isReviewed: boolean
- isAccepted: boolean
- response: String
Chat
- chatId: int
- senderId: int
- messageHistory: List<Message>
Message
- senderId: int
- date: DateTime
- messageBody: String
- messageId: int
Course
- courseId: int
- courseCode: String
- courseName: String
- semester: String
- instructor: List<Instructor>
- students: List<Student>
- university: University
- TAList: List<TA>
- tutorList: List<Tutor>
+ updateCourse(curCourseName: String, newCourseName: String, newCourseCode: String, TAs: List<TA>, graders: List<TA>, tutors: List<Tutor>, studentList: File)
+ extractStudents(studentsList: File)
CourseAnalytics
- courseAverage: double
- courseSD: double
- mostPopularQuestions: List<String>
- allSubmittedLabs: List<LabId><LabAssignments>
+ calculateCourseAverage(): double
+ calculateCourseSD(): double
+ findMostPopularQuestions(): List<String>
LabAnalytics
- labAverage: double
- labSD: double
- mostPopularQuestions: List<String>
- submittedLabs: List<LabAssignments>
+ calculateLabAverage(): double
+ calculateLabSD(): double
+ findMostPopularQuestions(): List<String>
GradingBot
- grade: int
- explanation: String
+ grade(assignment: File): int
+ assignmentPartsGrades(assignment: LabAssignment): List<part: String><grade: Int>
+ generateTestCase(labFile: LabAssignment): List<TestCase>
Section
- sectionNo: int
- courseId: int
- students: List<Student>
- labList: List<Lab>
LabAssignment
- labNo: int
- deadline: DateTime
- publishTime: DateTime
- labFile: File
- language: String
- version: String
- section: int
- testCases: List<TestCase>
- graders: List<User>
- assignmentParts: List<part: String><grade: Int>
- plagCheck: boolean
- plagRate: double
- plagCheckSettings: List<String>
- numberOfParts:int
- partialPoints: boolean
- codeAnalyzer: boolean
- lateSubmission: boolean
- latePolicyRate: double
- latePolicyTimeInterval: String
- autoGrading: boolean
- testCaseResult: boolean
+ setLabSettings(name: String, file: File, deadline: DateTime, partialPoints: boolean, allowCodeAnalyzer: boolean, graders: List<User>, lateSubmission: boolean, latePolicyType: String, latePolicyRate: double, testCases: List<TestCase>, autoGrading: boolean, publishTime: DateTime, testCaseResults: booelean, language: String, version: String, plagCheck: boolean, plagRate: double, plagCheckSettings: List<String>, section: int)
TestCaseRunner
+ runTestCase(testcase: TestCase, assignmet: File): boolean
TestCase
- input: String
- output: String

Fig. 3.5.3.1: Object and Class Diagram

Link of the class diagram is given below:

https://lucid.app/lucidchart/1fdf6e8b-b765-4ade-9546-24741f195589/edit?viewport_loc=-2370%2C-2177%2C5755%2C3110%2C0_0&invitationId=inv_300b5ce8-8071-4322-abc5-9af2e1a4aa53

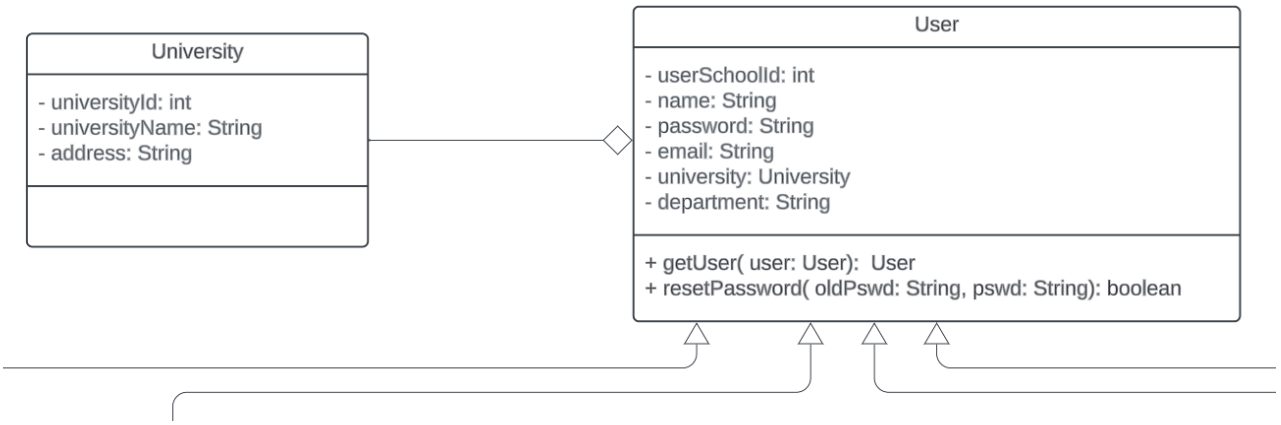

Fig. 3.5.3.2: User and University Classes

User Class: User class is the parent of all user types (Student, Instructor, Tutor, TA) and includes common properties such as name, school ID, university, email, department, and password.

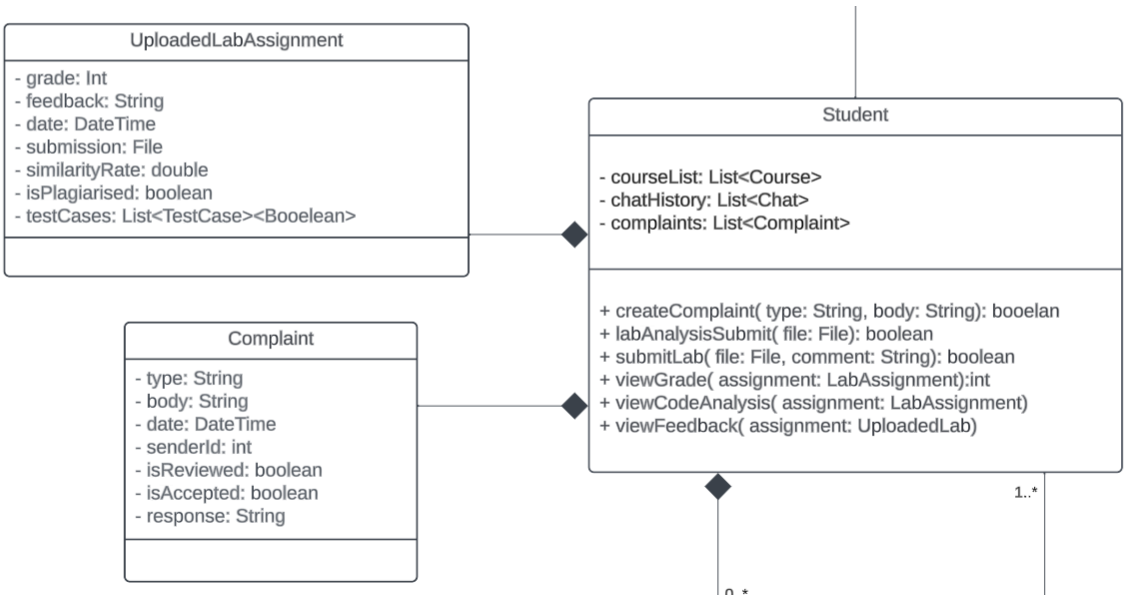

Fig. 3.5.3.3: Student, UploadLabAssignment, and Complaint Classes

Student Class: This class represents the student user type. Each student has a course list, chat history, and complaints that they have created. There are several functions specific to student user types. createComplaint function is used to create a complaint, which takes the complaint type and body as parameters. labAnalysisSubmit allows a student to submit a lab file to be analyzed, submitLab to submit the lab for grading, viewGrade to view the grade of the submitted assignment/lab, viewCodeAnalysis to view the code analysis, viewFeedback to view the feedback of the graded assignment/lab.

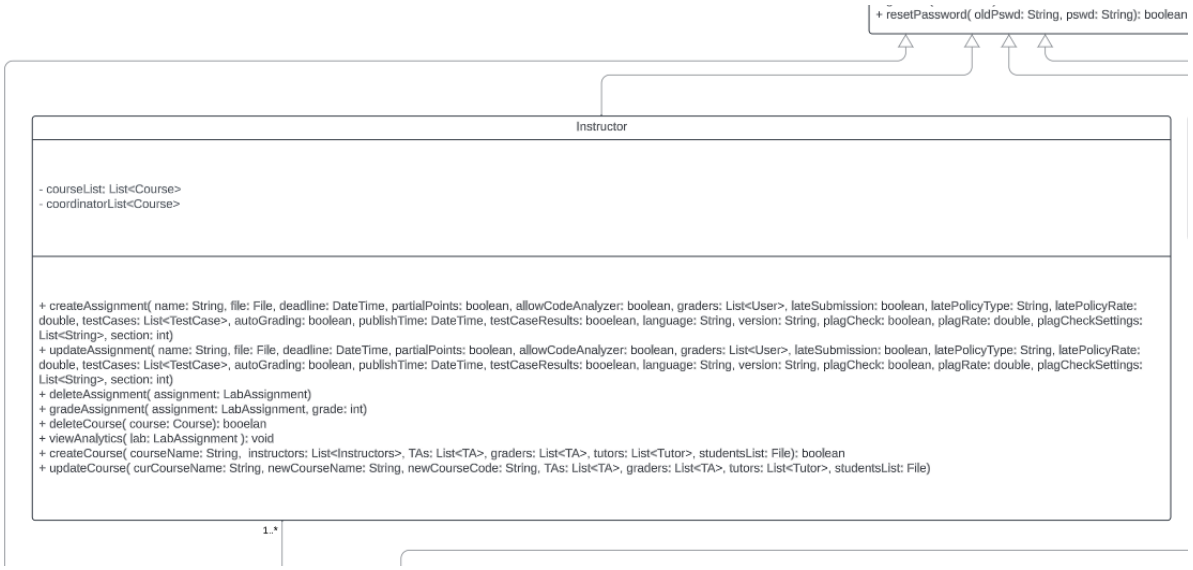

Fig. 3.5.3.4: Instructor Class

Instructor Class: This class represents the instructor user type. An instructor has a list of courses he/she gives and a list of courses that the instructor is a coordinator of, if any. By using createAssignment function, instructors can create a lab assignment by specifying lab properties as parameters, with updateAssignment, they can update a created lab assignment's properties, with deleteAssignment, they can delete an already existing lab assignment, with gradeAssignment, they can grade an assignment that is submitted by a student, with deleteAssignment, they can delete an existing course, with viewAnalytics, they can view the analytics of a lab or course, with createCourse, they can create a course by specifying the course properties as parameters, and with updateCourse, they can update an existing course.

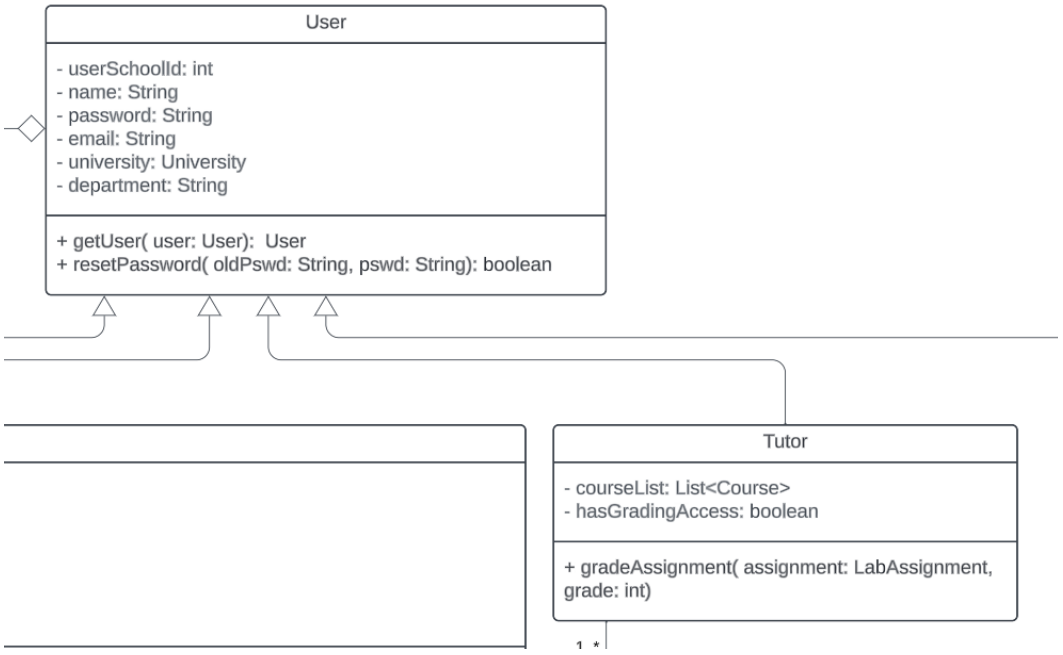

Fig. 3.5.3.5: User and Tutor Classes

Tutor Class: This class represents the tutor user type. A tutor has a course list and a property to indicate whether that tutor has grading access or not. If a tutor has access to grade an

assignment, gradeAssignment is used to grade the assignment is used with the submitted lab assignment, and the grade is specified as a parameter to the function.

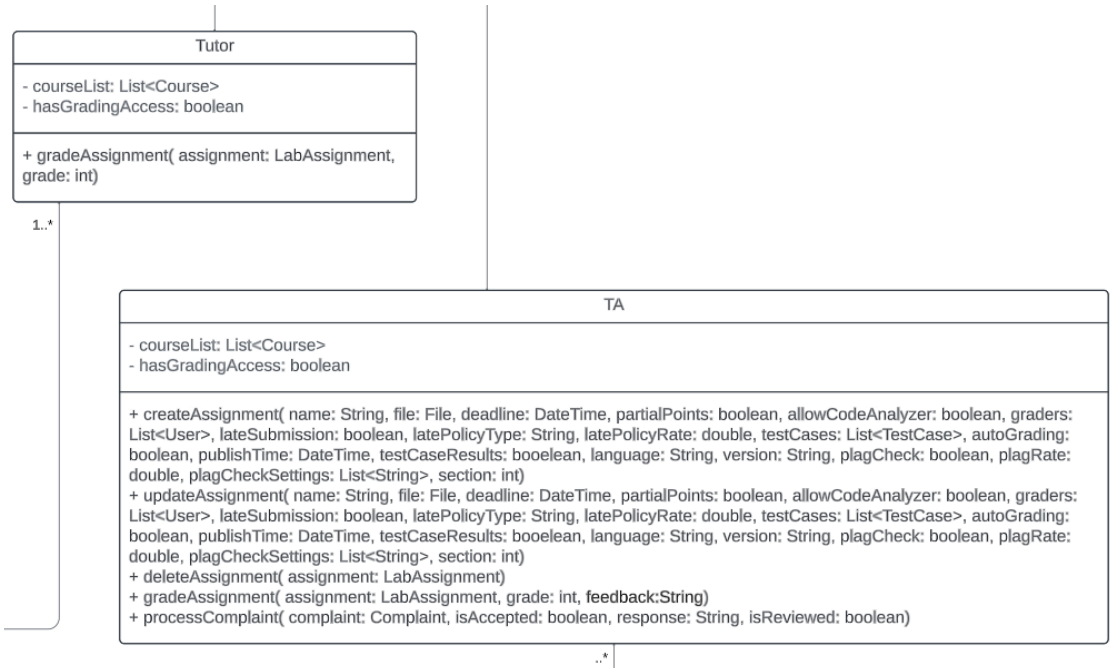

Fig. 3.5.3.6: Tutor and TA Classes

TA Class: This class represents the TA user type. A TA has a course list and a property to indicate whether that TA has grading access or not. If a TA has access to grade an assignment, the gradeAssignment function is used to grade the assignment, with the submitted lab assignment, and the grade is specified as a parameter to the function. Furthermore, a TA can create a lab assignment by specifying the lab properties as parameters to the createAssignment function. A TA can also update an existing lab assignment with updateAssignment function and delete an existing lab assignment with deleteAssignment function. Additionally, using processComplaint, a TA can view a complaint, reject or accept, and give response.

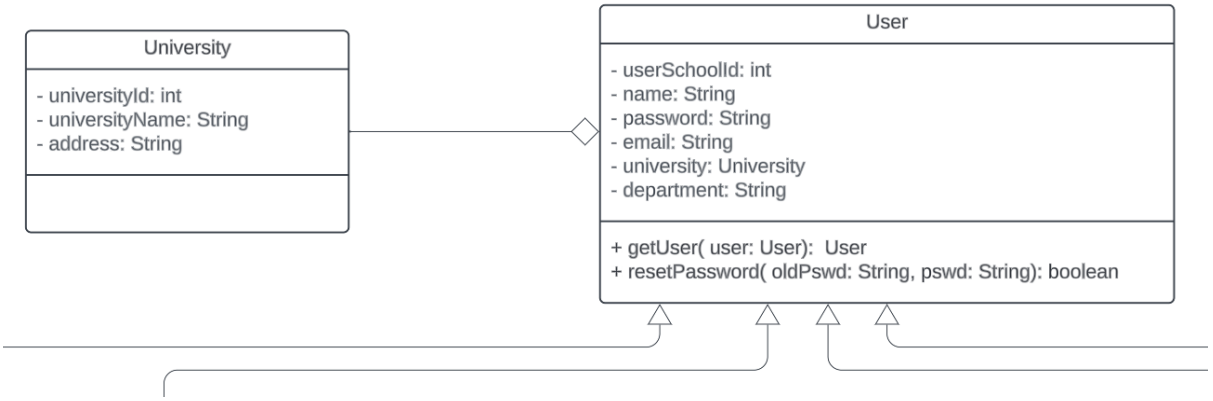

Fig. 3.5.3.7: User and University Classes

University Class: Each user has a university as one of their properties. A university object has a name, id, and address.

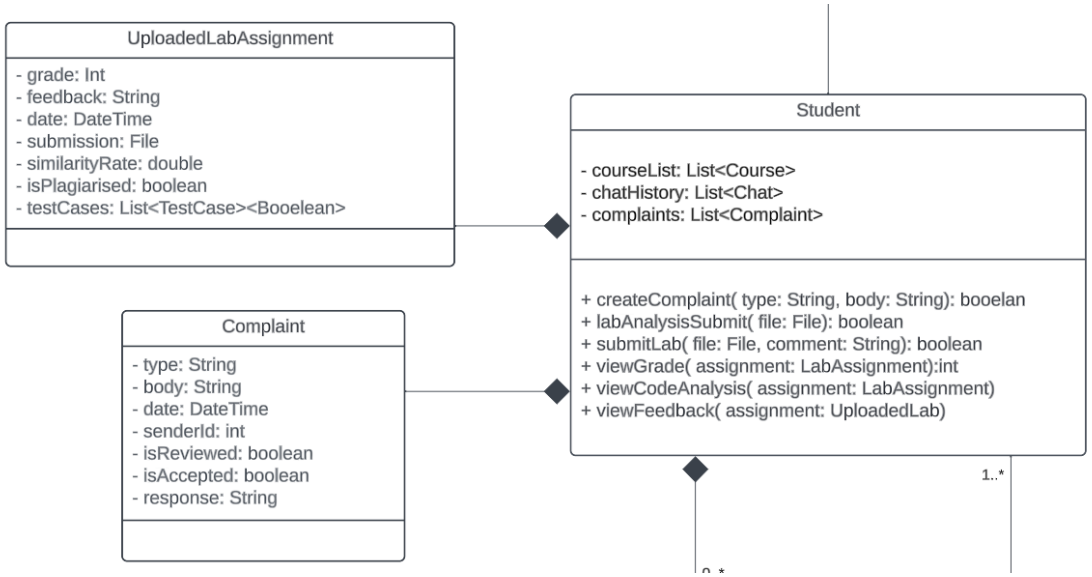

Fig. 3.5.3.8: Student, UploadLabAssignment, and Complaint Classes

UploadedLabAssignment Class: Students have uploaded lab assignments if they have submitted a lab assignment for grading. A lab assignment has its grade, feedback from the grader, date of submission, similarity rate, plagiarism information, and list of test cases with their results.

Complaint Class: Complaint class represents a complaint created by a student. A complaint object has a type property, which indicates whether the complaint is made for a grade of an assignment or for the chatbot. Additionally, a complaint has a body, creation date, sender id, two boolean variables to indicate whether the complaint is reviewed and accepted, and a response from the reviewer.

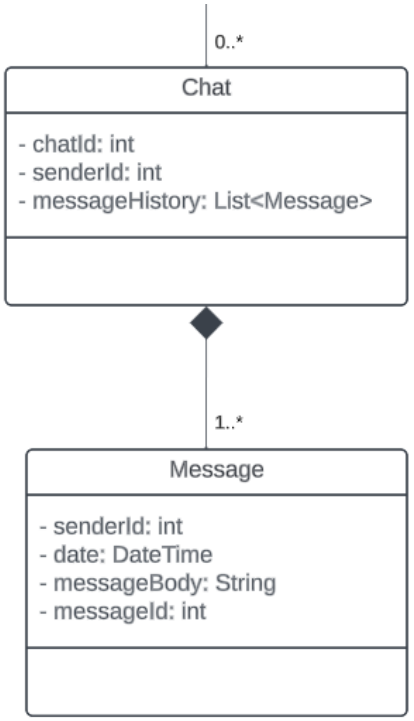

Fig. 3.5.3.9: Chat and Message Classes

Chat Class: Every student has a list of chats, which are the chats made with the chatbot. A chat represents a single chat and a list of chats creates a chat history. A chat has its id, sender id, and a message history, which is a list of messages.

Message Class: A message is an object which comes together to create a single chat. A message has a sender id, date of sent, message body, and a message id.

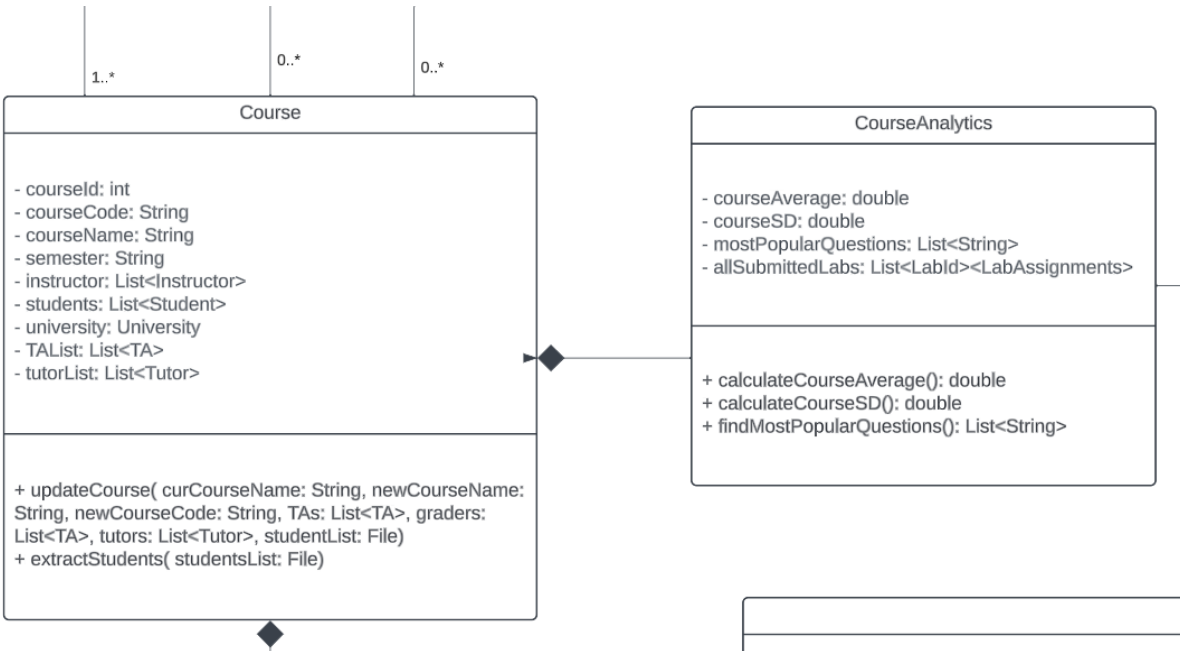

Fig. 3.5.3.10: Course and CourseAnalytics Classes

Course Class: The course class represents a course created by an instructor. A has a course id, course code, course name, semester, instructor list, student list, TA list, tutor list, and university. With the updateCourse function, properties of a course can be updated. With the extractStudents function, a list of students enrolled in the course can be returned.

CourseAnalytics Class: CourseAnalystics class is created to present an analysis of a particular course. The class has properties of course average, course standard deviation, list of most popular questions asked to the chatbot, list of all submitted labs by the students of that particular course. calculateCourseAverage calculates the assignment grade average of the course. calculateCourseSD function calculates the assignment standard deviation average of the course. findMostPopularQuestions return a list of most popular questions asked in the whole course.

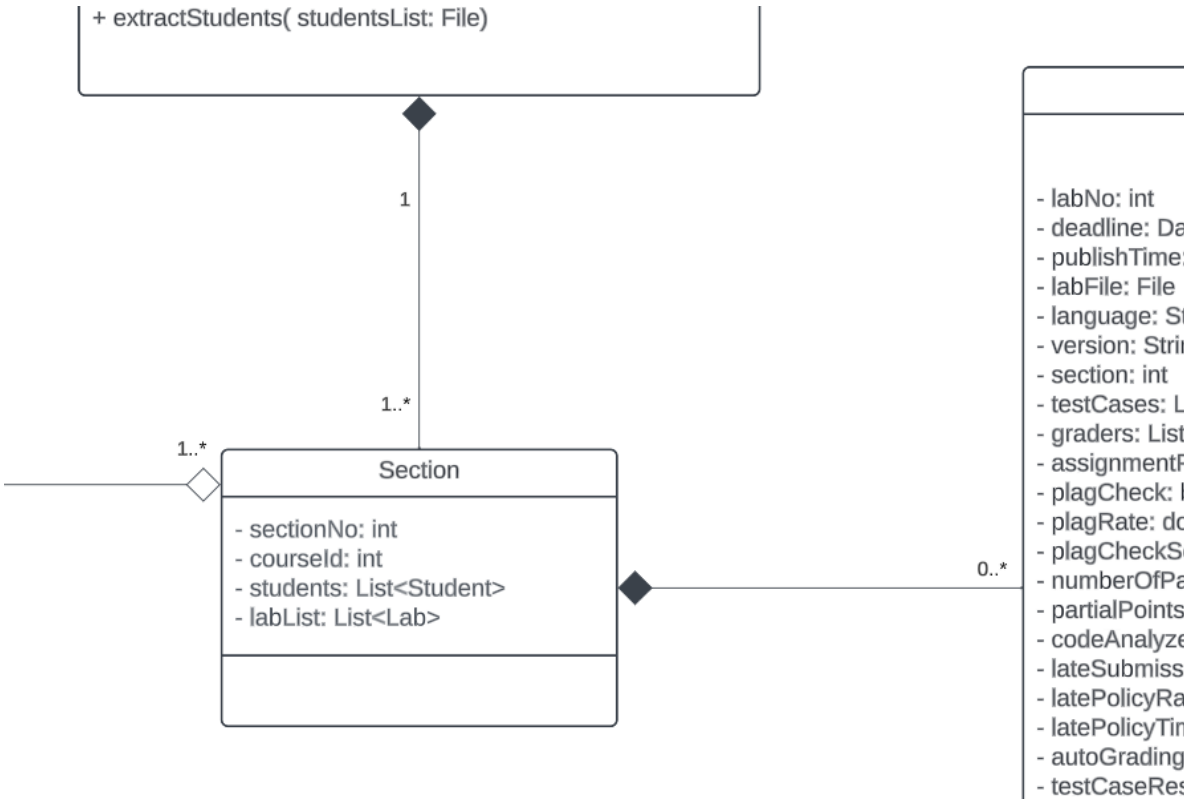

Fig. 3.5.3.11: Section Class

Section Class: Each course has a number of sections and each section object has a section no, course id it belongs to, list of students enrolled in that section, and a list of labs created for that section.

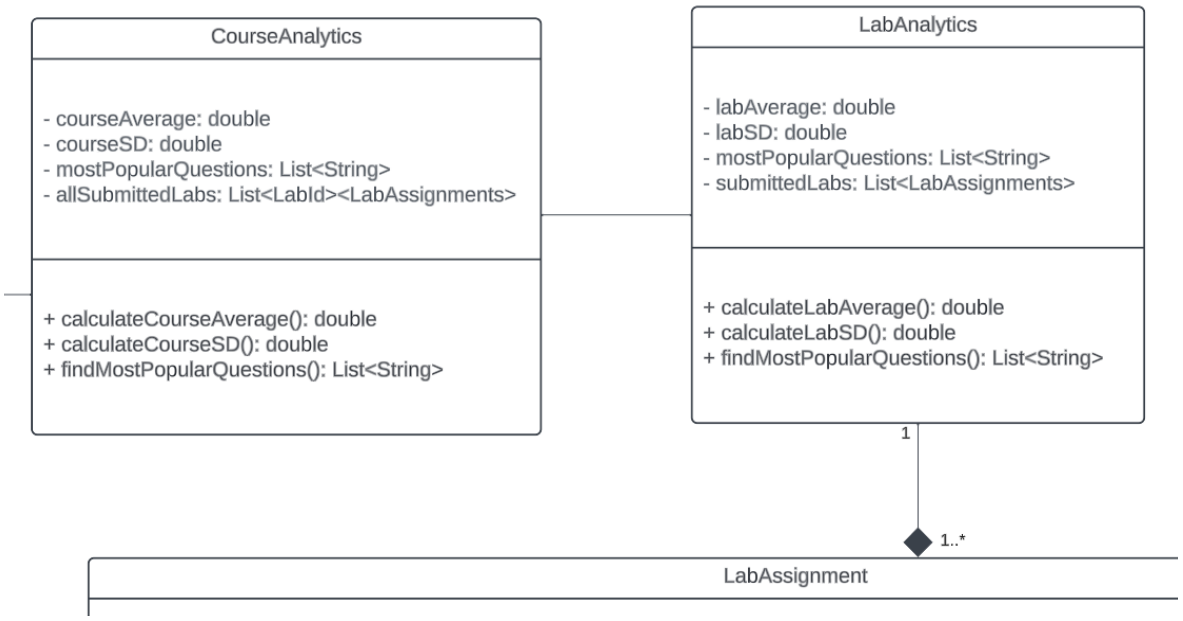

Fig. 3.5.3.12: LabAnalytics and CourseAnalytics Classes

LabAnalytics Class: LabAnalystics class is created to present an analysis of a particular course. The class has properties of lab grade average, lab standard deviation, list of most popular questions asked to the chatbot, list of all submitted labs by the students of that particular lab. calculateLabAverage calculates the assignment grade average of the lab. calculateLabSD function calculates the assignment standard deviation average of the lab. findMostPopularQuestions return a list of most popular questions asked in the whole lab.

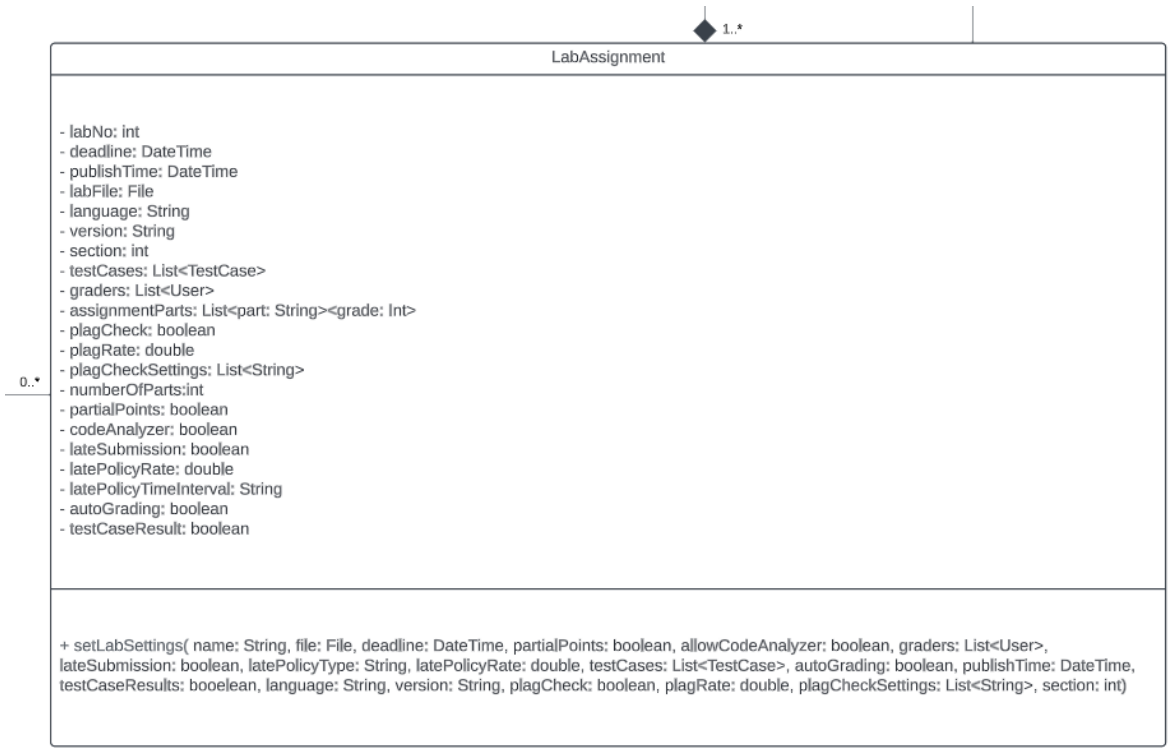

Fig. 3.5.3.13: LabAssignment Class

LabAssignment Class: LabAssignment class represents an assignment created by the TA or an instructor for students to work on. It has the properties of lab number int, deadline, assignment publishment time, assignment file, programming language of the assignment, programming language version, section number, list of test cases, list of graders, list of assignment parts and corresponding grades, plagCheck to indicate whether plagiarism check will be done after the submission, accepted plagiarism rate, decide on whether plagiarism check will be done over a single section or the whole course sections, number of parts, partialPoints to indicate if partial points are given, codeAnalyzer to indicate whether students will have access to the code analyzer before submission, lateSubmission to indicate whether late submission is allowed, late policy grade reduction rate, latePolicyTimeInterval, autoGrading to indicate whether auto grading of the assignment by the system is allowed, testCaseResult to indicate whether students will be able to see test case results of their

assignments before the submission. setLabSettings function is used to set the properties of the lab.

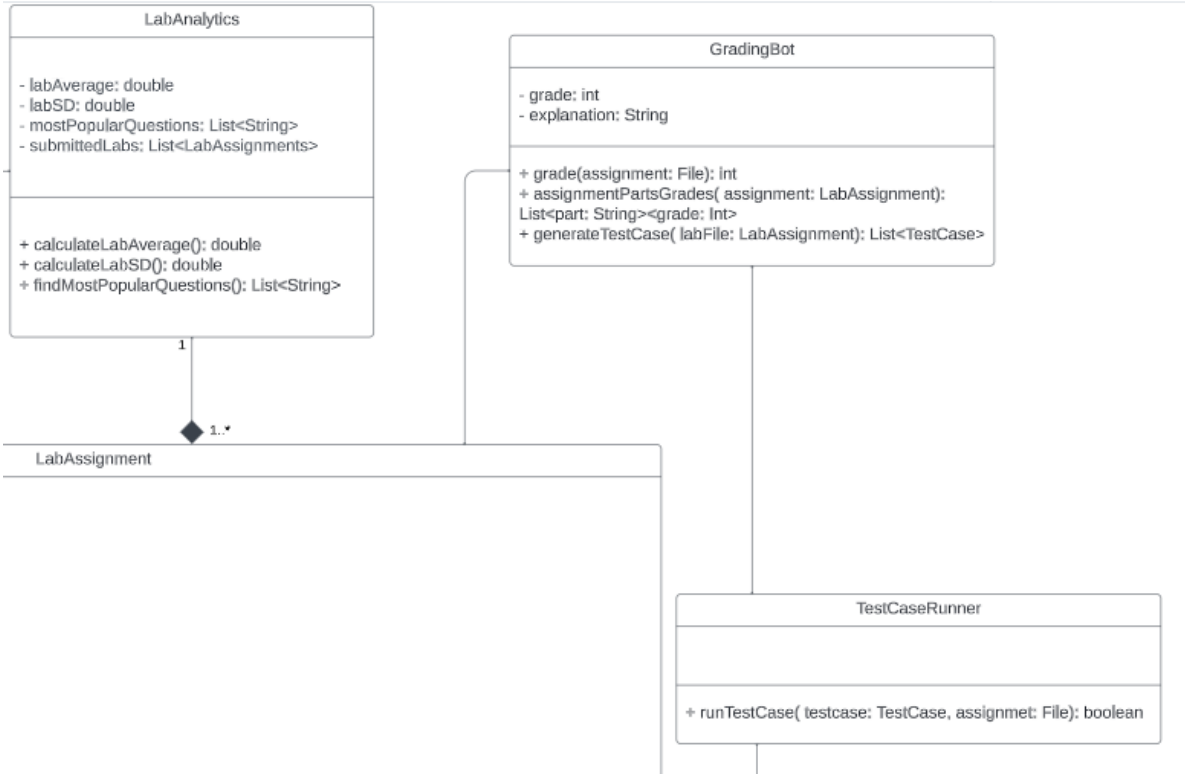

Fig. 3.5.3.14: LabAssignment, LabAnalystics, GradingBot, and TestCaseRunner Classes

GradingBot Class: This class runs test cases on an assignment and calculates the final grade of a submitted assignment. It has the properties of grade and the explanation for that grade. grade function is where the grade is calculated. assignmentGrade takes the decided grade, by the instructor, of the corresponding assignment. The class also generates test cases if automatic test case generation is enabled.

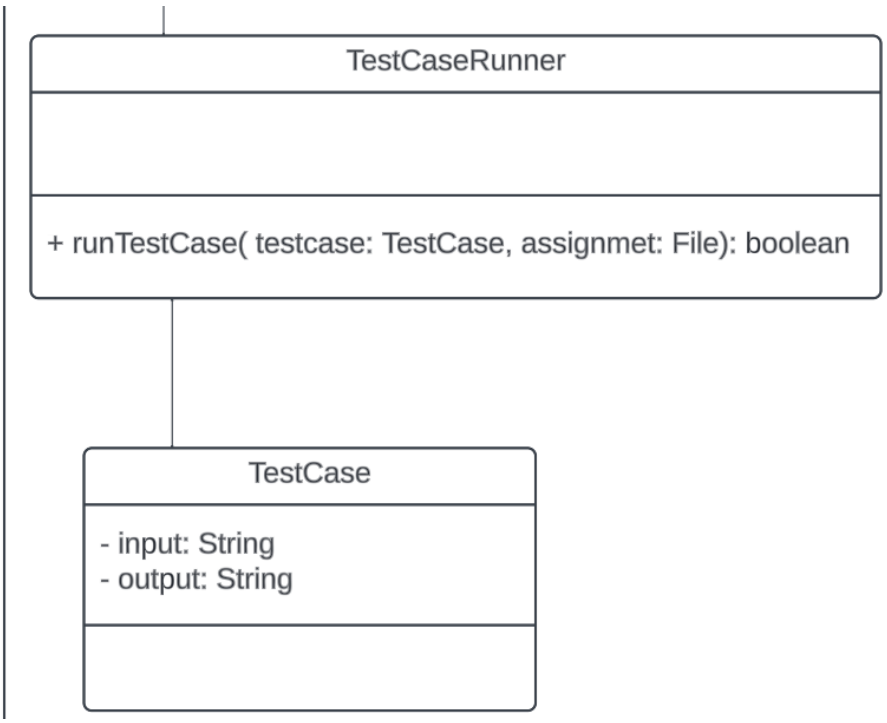

Fig. 3.5.3.15: TestCaseRunner and TestCase Classes

TestCaseRunner Class: This class only runs test cases on a given assignment with given test cases using runTestCase function.

TestCase Class: TestCase class represents a single test case. It has properties of input, which will be given to the submitted assignment code as input and an output, which will be compared with the actual output of the submitted assignment.

3.5.4. Dynamic Models

3.5.4.1. Activity Diagrams

Lab Process Activity Diagram

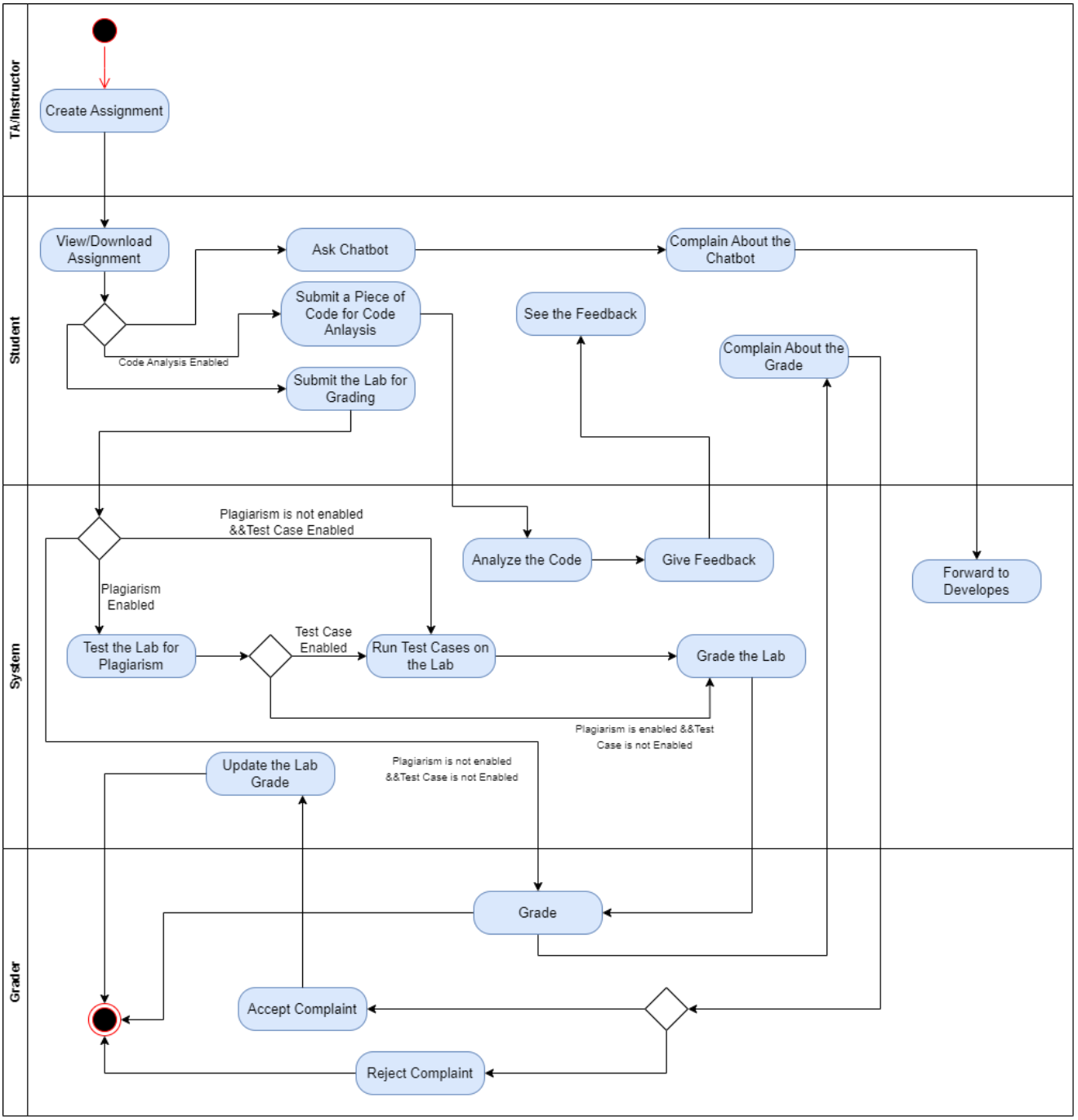

Fig. 3.5.4.1.1: Lab Process Activity Diagram

The above activity diagram displays the whole process of a lab assignment creation by a TA or instructor, submission by the student, processment by the system, and the final grading by the grader.

3.5.4.2. State Diagrams

Uploaded Lab State Diagram

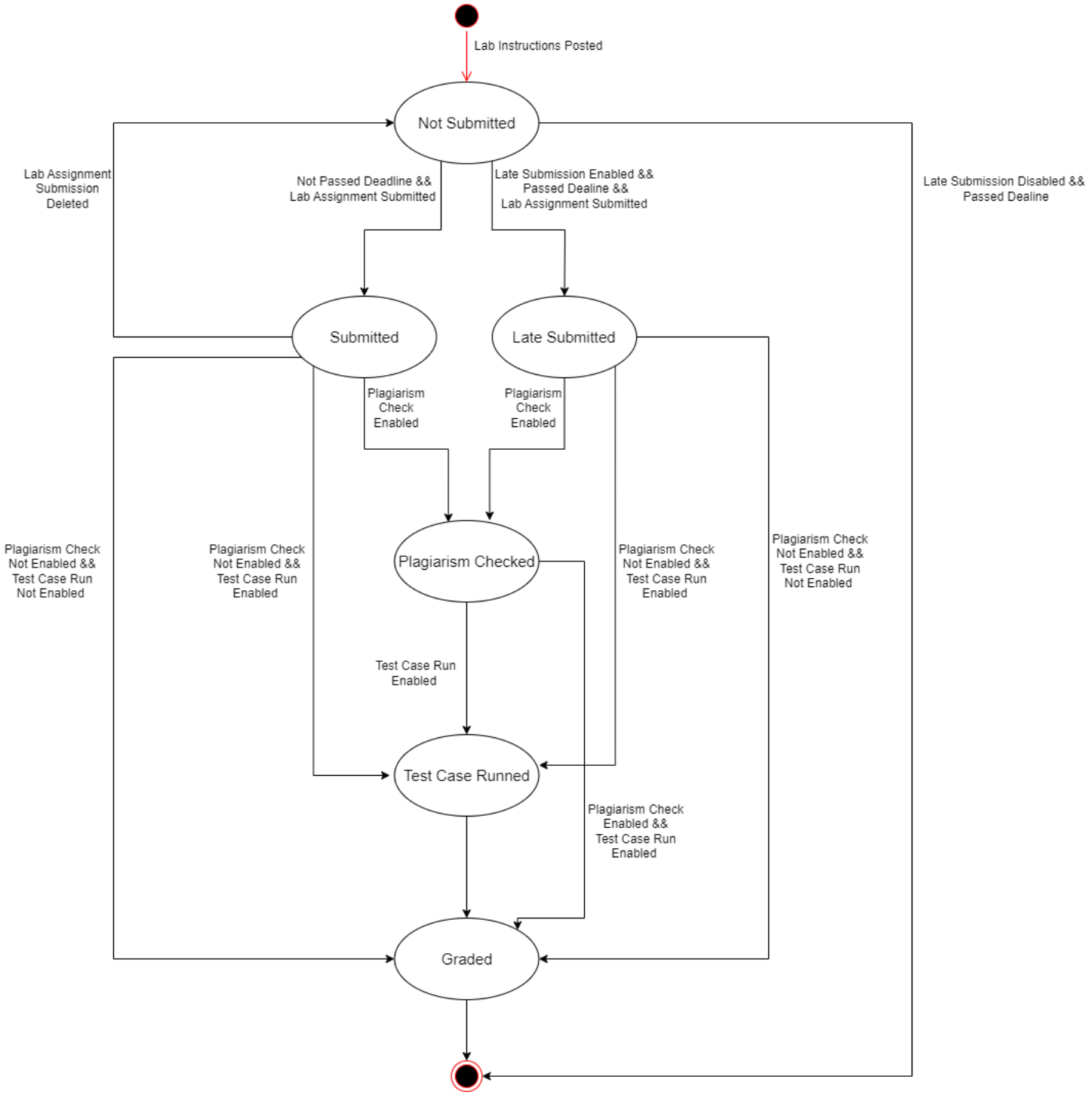

Fig. 3.5.4.2.1: Uploaded Lab State Diagram

The above state diagram displays the states that a lab assignment is going through, which is submitted by the student for grading.

Complaint State Diagram

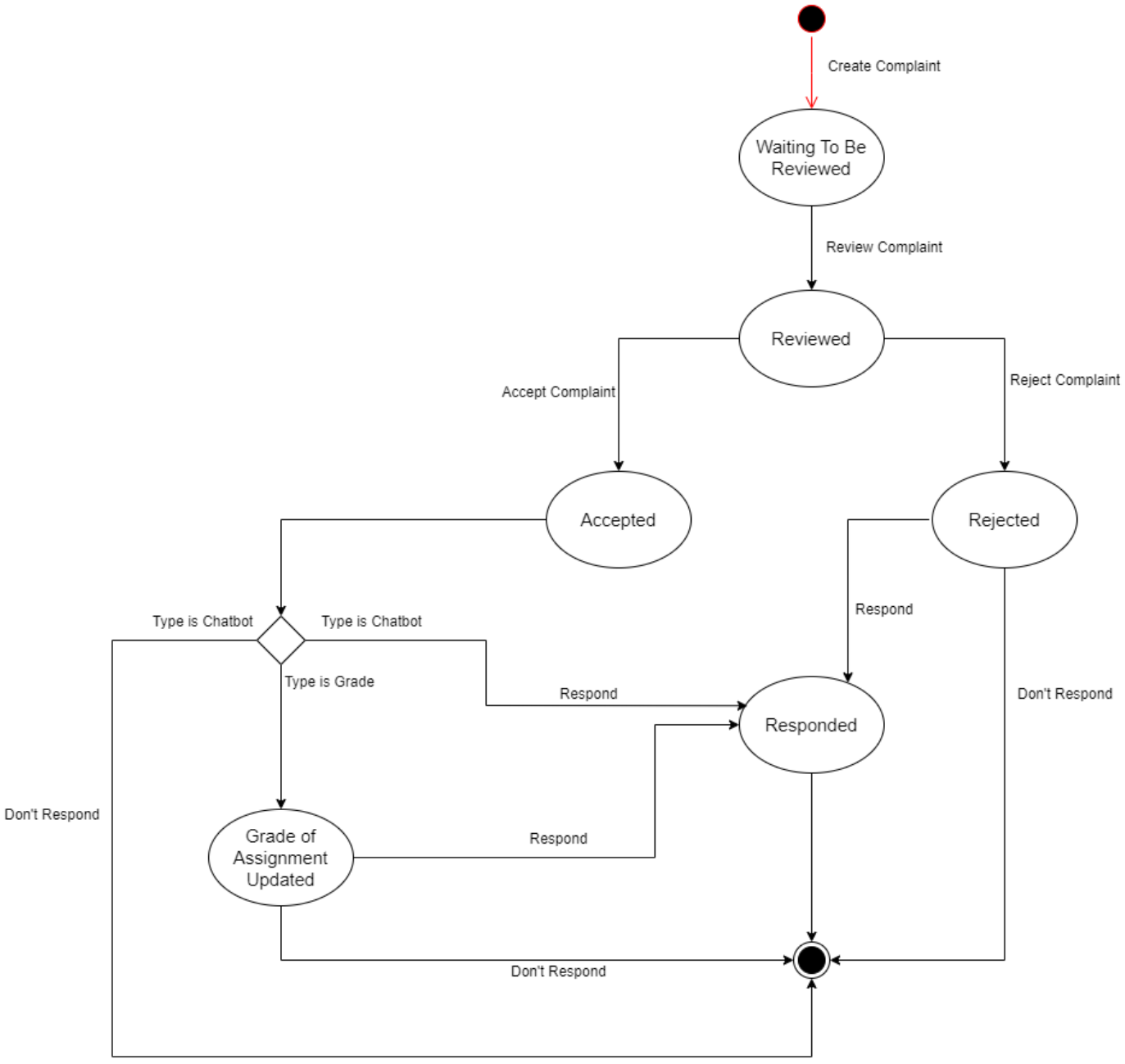

Fig. 3.5.4.2.2: Complaint State Diagram

The above state diagram displays the states that a complaint is going through once it is created by the Student. A complaint can be created for two purposes, one for complaining about the lab grade, and the other is for complaining about the chatbot.

3.5.4.3. Sequence Diagrams

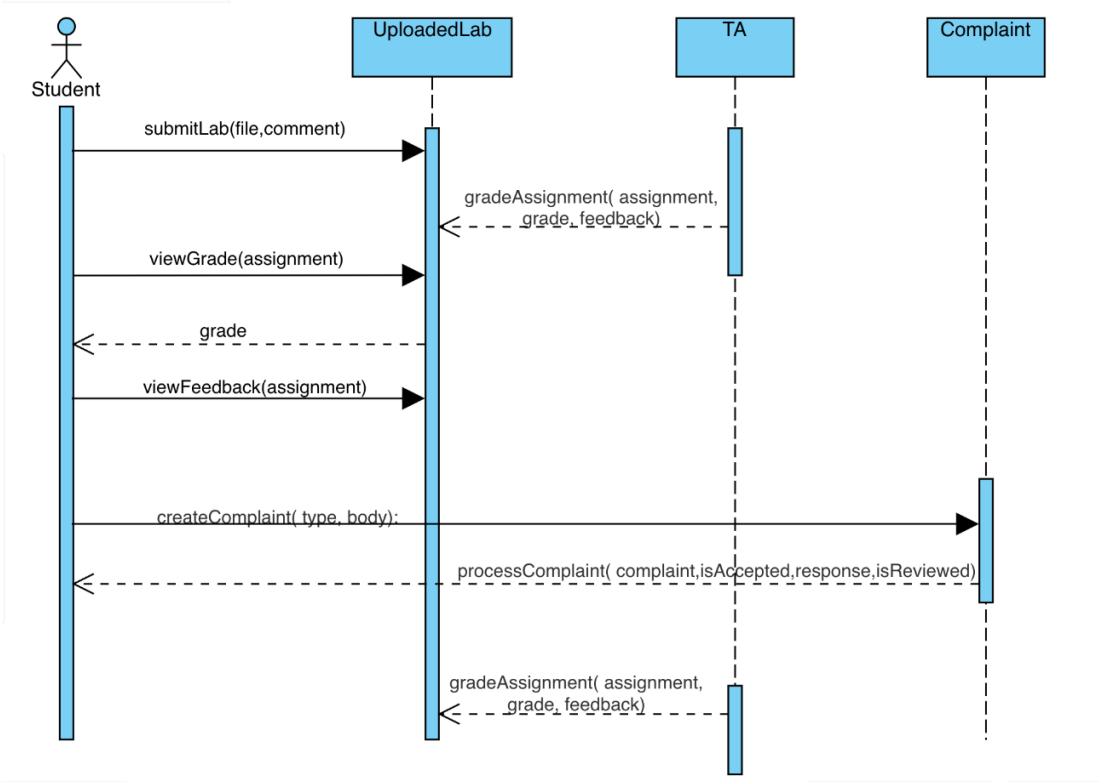

Fig. 3.5.4.3.1: Lab Grade Complaint Sequence Diagram

This diagram illustrates the process that begins when a student submits their lab assignment. Following the submission, a TA evaluates the work, assigns a grade, and provides feedback. The student then reviews this grade and feedback. If the student disagrees with the evaluation, they have the option to file a complaint. This complaint is then reviewed, and if deemed valid, the TA re-evaluates and, if necessary, adjusts the lab grade accordingly.

3.5.5. User Interface

3.5.5.1. Common Interfaces

Common interfaces are available for any user type. These pages do not show differences between user types.

Landing Page

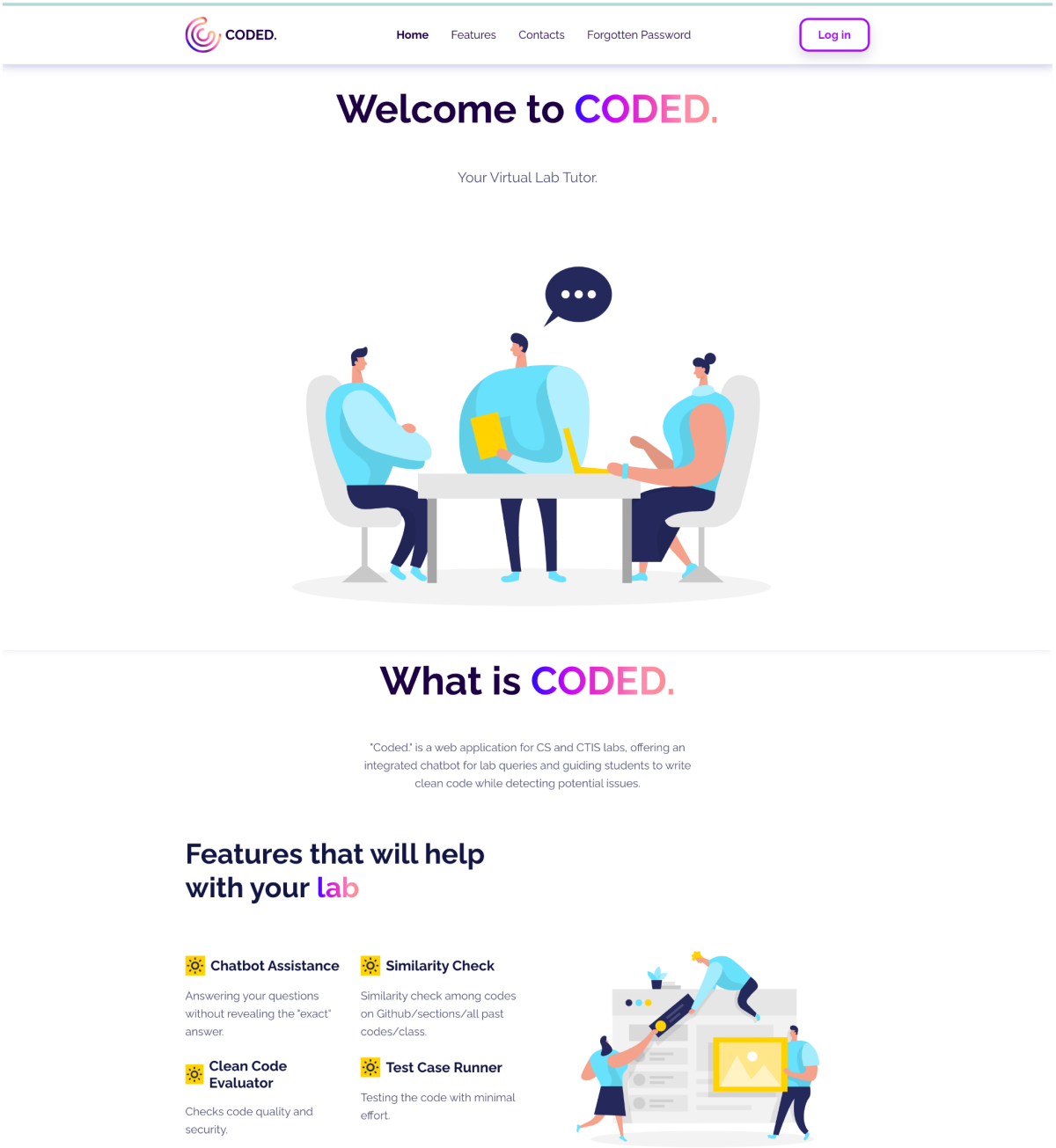

Fig. 3.5.5.1.1.: Landing Page

This is the first page a user is faced with when visiting CODED. Through this page, users can see the features of the application, go to the log in, contact us, and forgotten password pages through the navbar at the top.

Log In Page

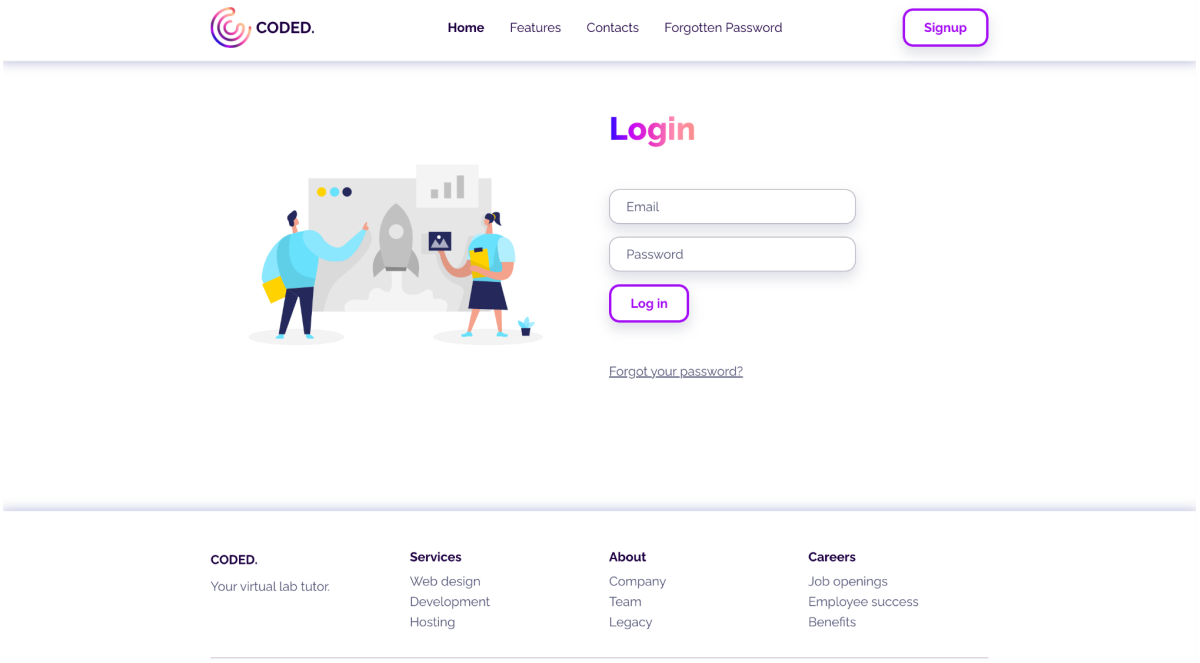

Fig. 3.5.5.1.2: Log In Page

Users can log in to the system if they already have an account by providing their registered email addresses and passwords. Users can navigate to the forgotten password page if they have forgotten their password. Additionally, they can go to the landing page, forgot password, and sign up from the navbar at the top.

Signup Page

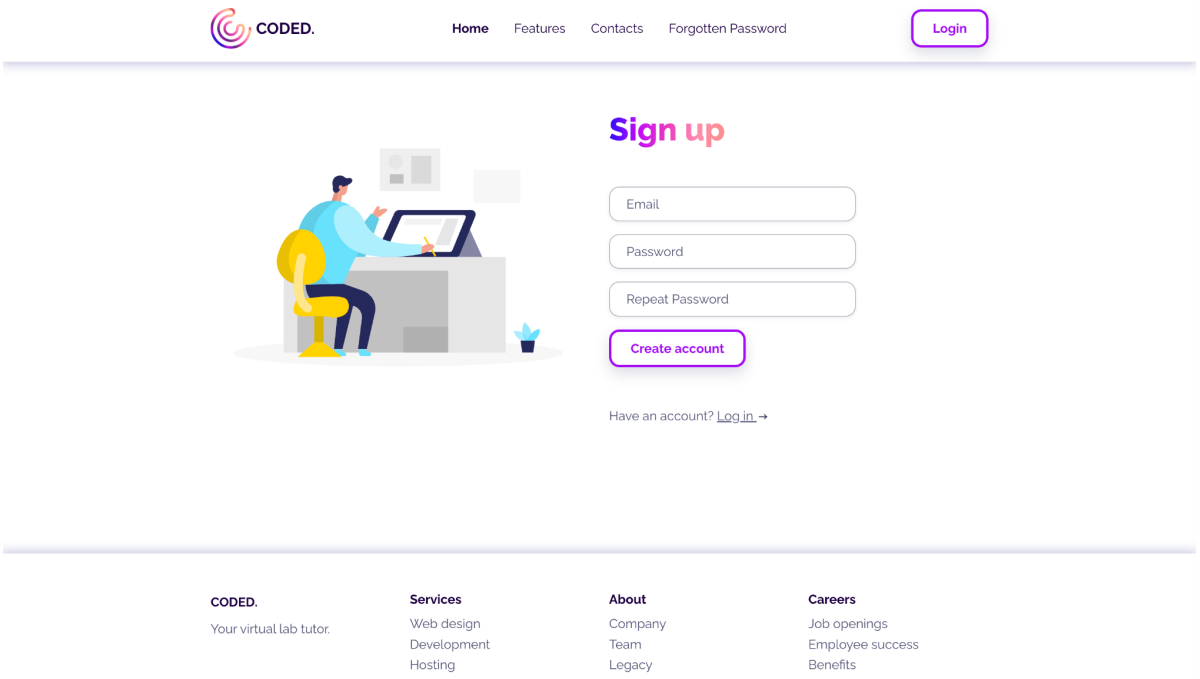

Fig. 3.5.5.1.3: Signup Page

This page allows a user to sign up to the system. Although any user type can reach the page, only instructor sign up is allowed. Instructors can sign up for the system by providing their mail addresses and passwords.

Forgotten Password Page

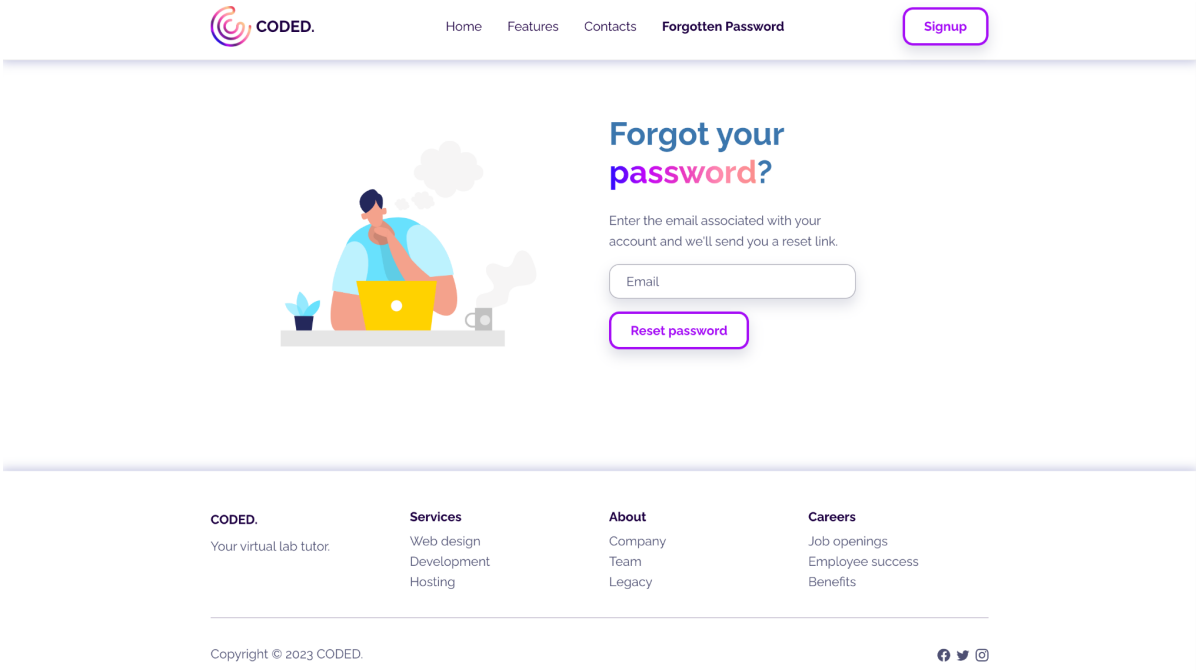

Fig. 3.5.5.1.4: Forgotten Password Page

This page allows users to reset their passwords through providing their registered email addresses. From the navbar at the top, users can reach the Landing page, features, contact and sign up pages.

3.5.5.2. Student Interfaces

Student Dashboard

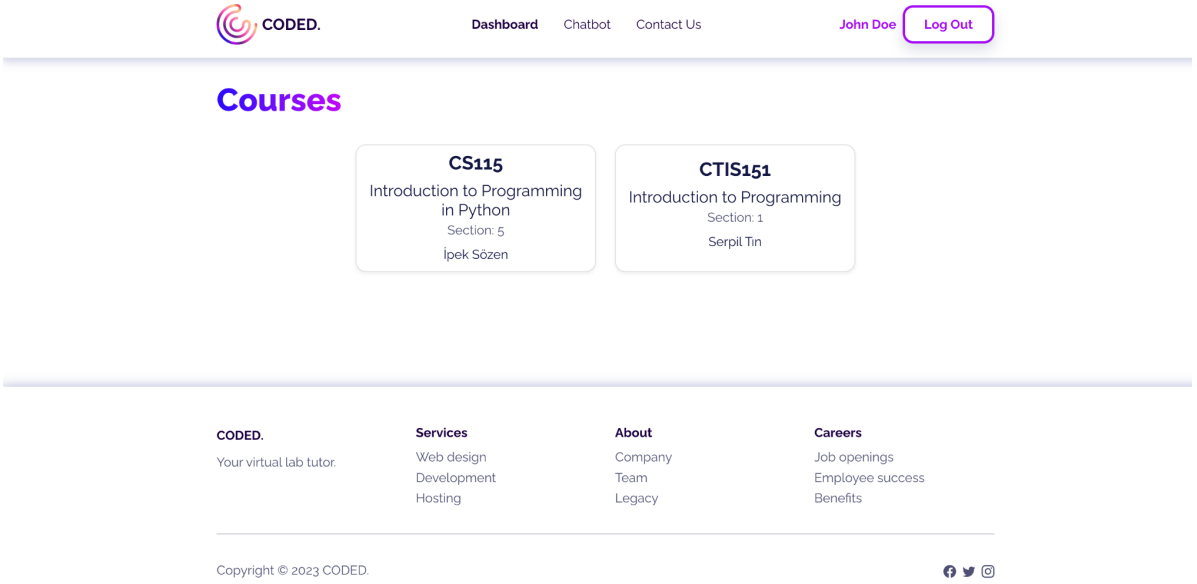

Fig. 3.5.5.2.1: Student Dashboard Page

When a student logs into the website, they first see the student dashboard page. Through the dashboard they can view the classes that they take and navigate to the main pages of their classes from the ‘Courses’ menu.

Lab Page

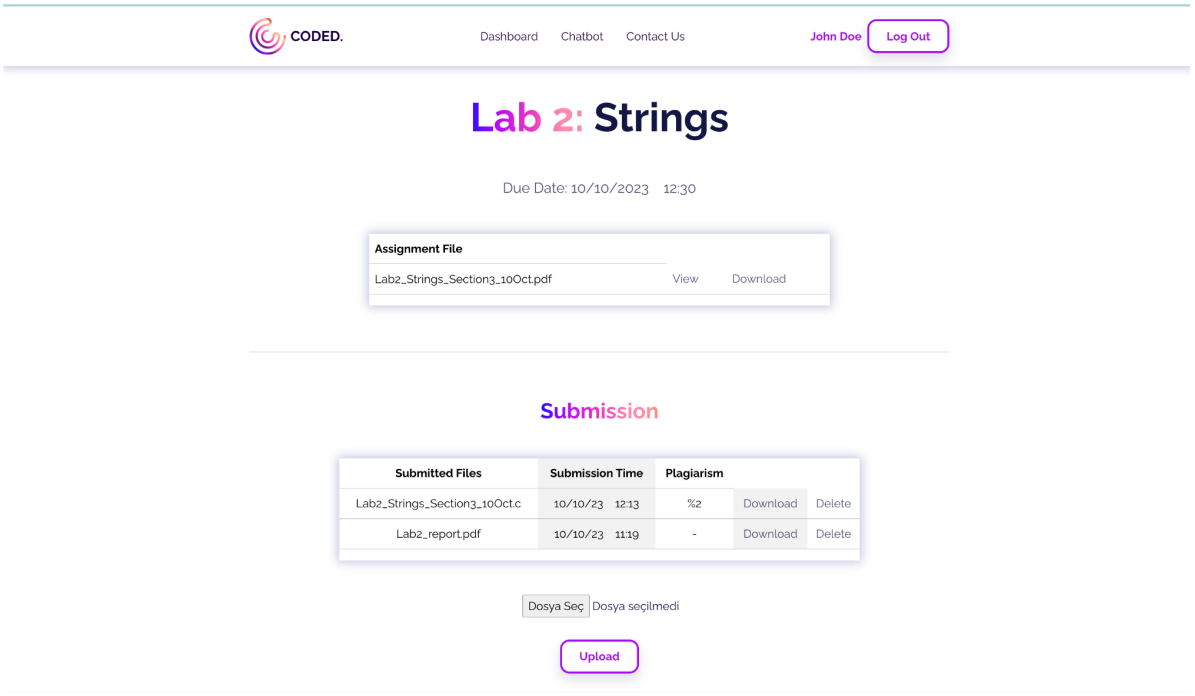

Fig. 3.5.5.2.2: Lab Page

Through the Lab Page, students can view or download the lab assignment uploaded by the TA or the instructor. They can also submit multiple lab files using the upload button after choosing files.

Published Assignments Page

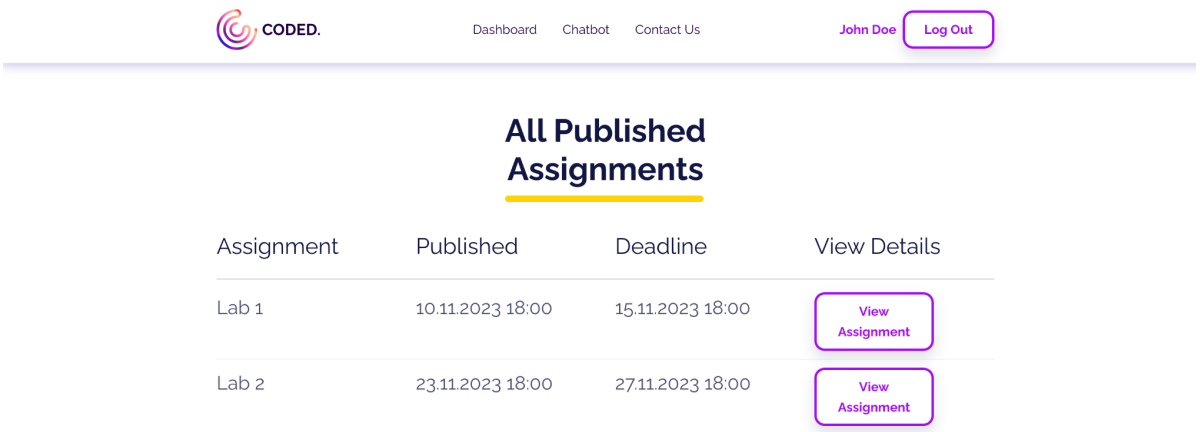

Fig. 3.5.5.2.3: Published Assignments Page

The student will be able to see a list of all uploaded labs and their submissions through this page. The student will be directed to an individual lab assignment and its submission after clicking on the respective button.

Uploaded Lab Page

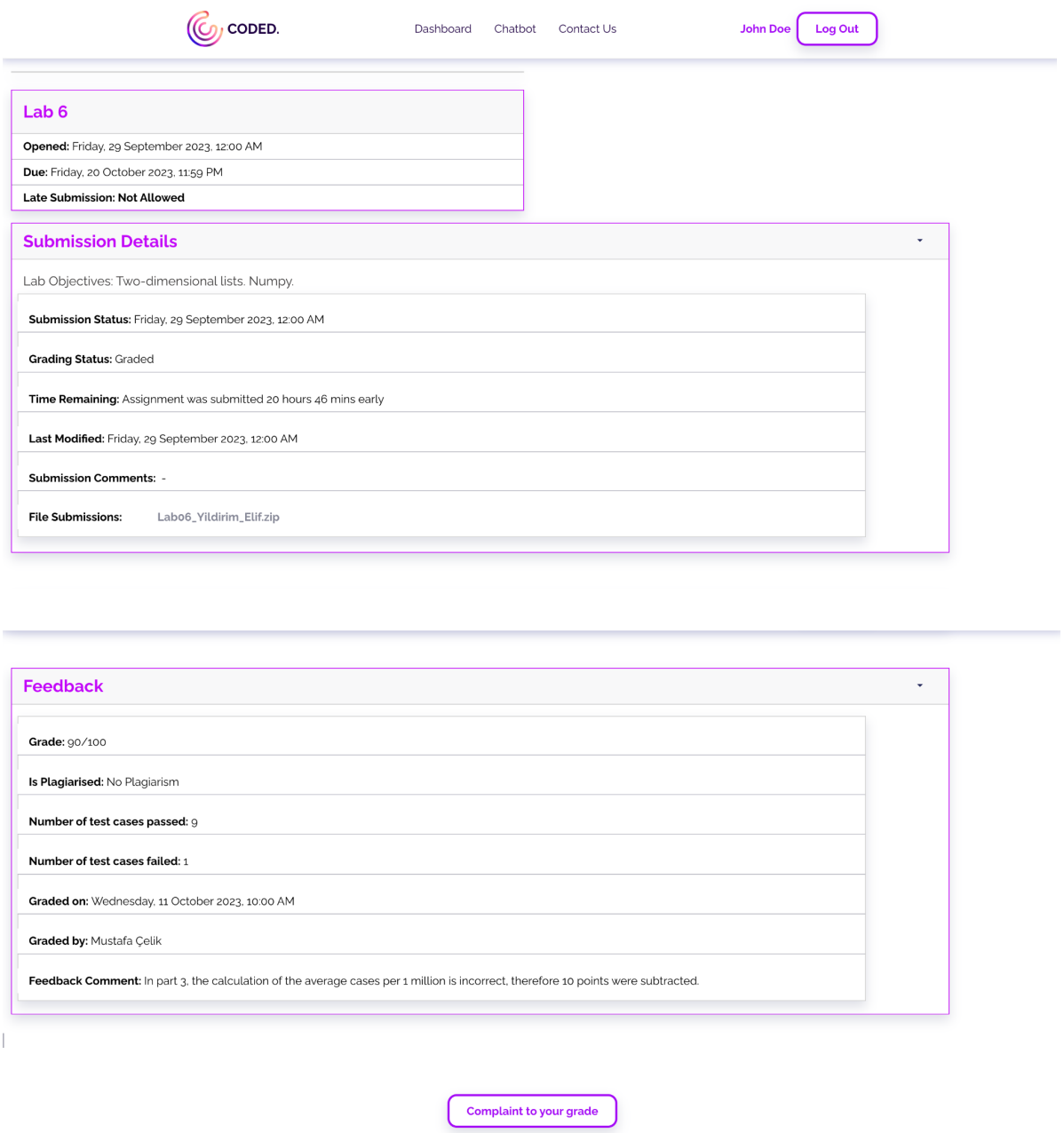

Fig. 3.5.5.2.4, Fig. 3.5.5.2.5: Uploaded Lab Page

On this page, users can access detailed information about each lab submission. This includes not only the basic details available on the pre-upload lab page – such as submission time, late submission policies, and other relevant submission data – but also comprehensive post-upload information. Users can view the lab's grade, the identity of the grader, any

feedback provided, the plagiarism status, and the number of test cases passed or failed. Additionally, there is an option for students to object to their grade, allowing them to attach a detailed explanation to their objection

Code Analysis Upload Page

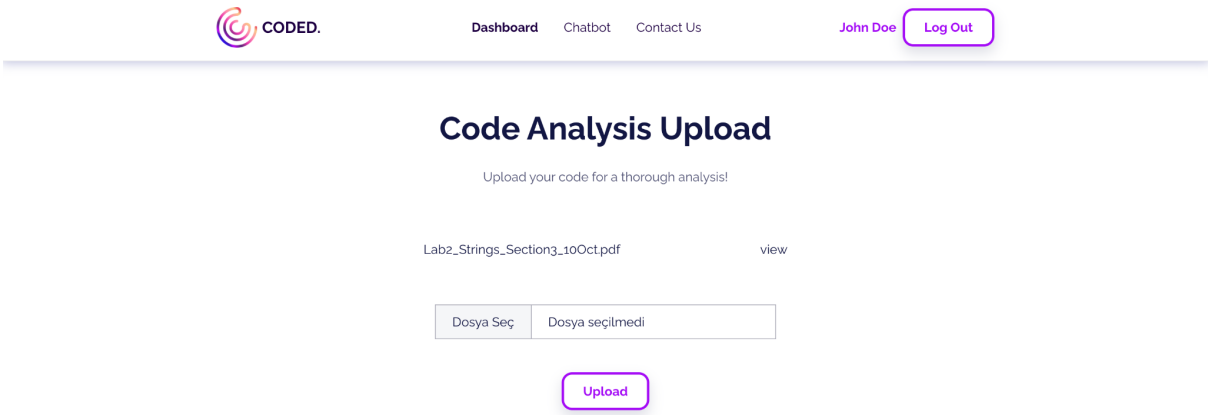

Fig. 3.5.5.2.6: Code Analysis Upload Page

On this page, each student user can upload their code on the application for a code analysis. The student can upload multiple files.

Code Analysis Page

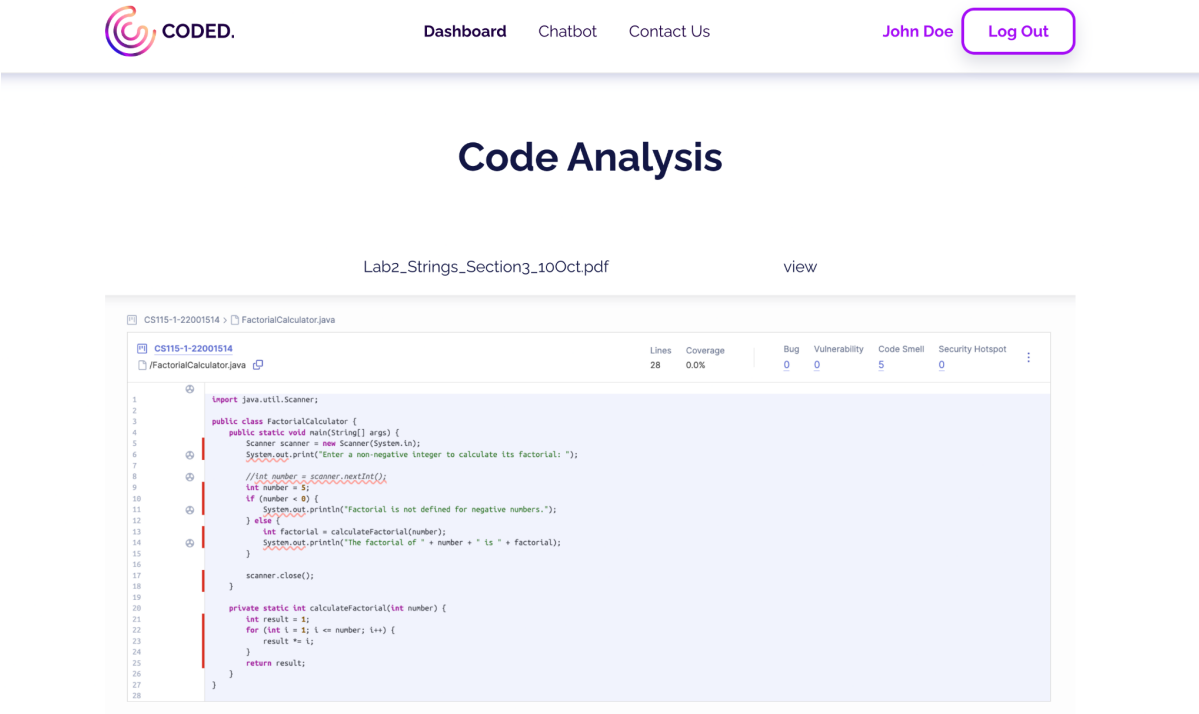

Fig. 3.5.5.2.7: Code Analysis Page

After uploading the code, the page will reload and the code analysis will be shown on the page as it can be seen. The code will show all code smells, bugs, vulnerabilities, and security problems on the code. The user will be individually guided to fix their code based on this feedback.

Profile Page

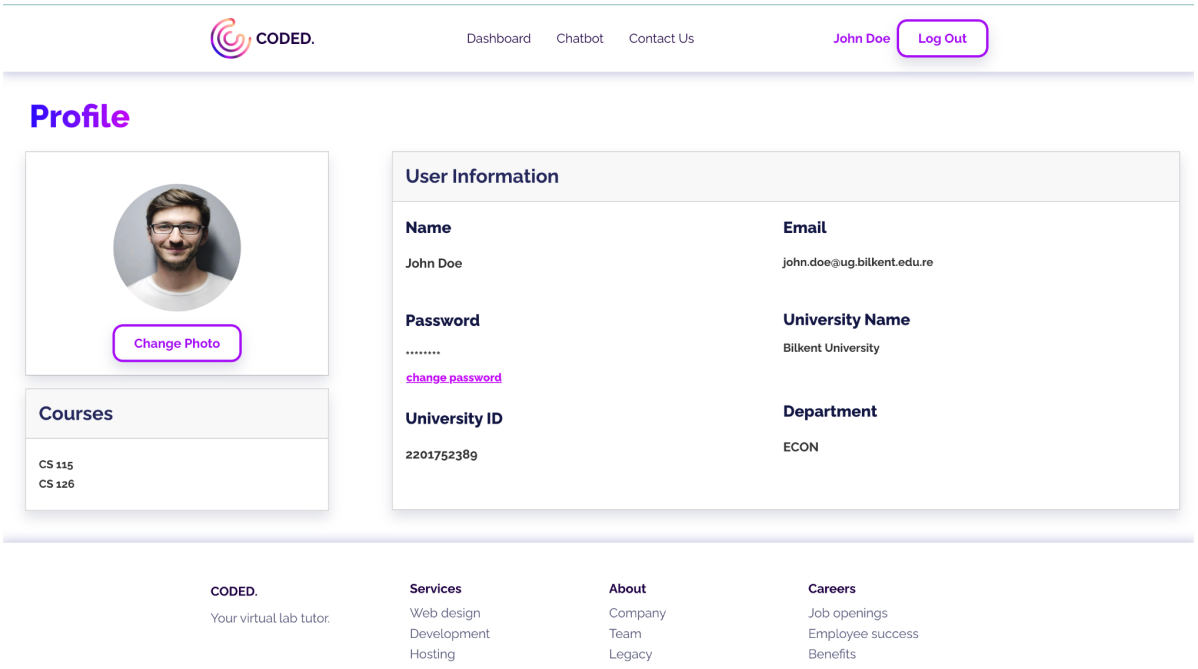

Fig. 3.5.5.2.8: Profile Page

On this page, essential details such as the student's name, surname, university ID, and department are displayed. Students can change their password here as well. Additionally, the page shows the names of the courses currently being taken by the student. The profile picture is also displayed and can be updated from this page.

Chatbot Page

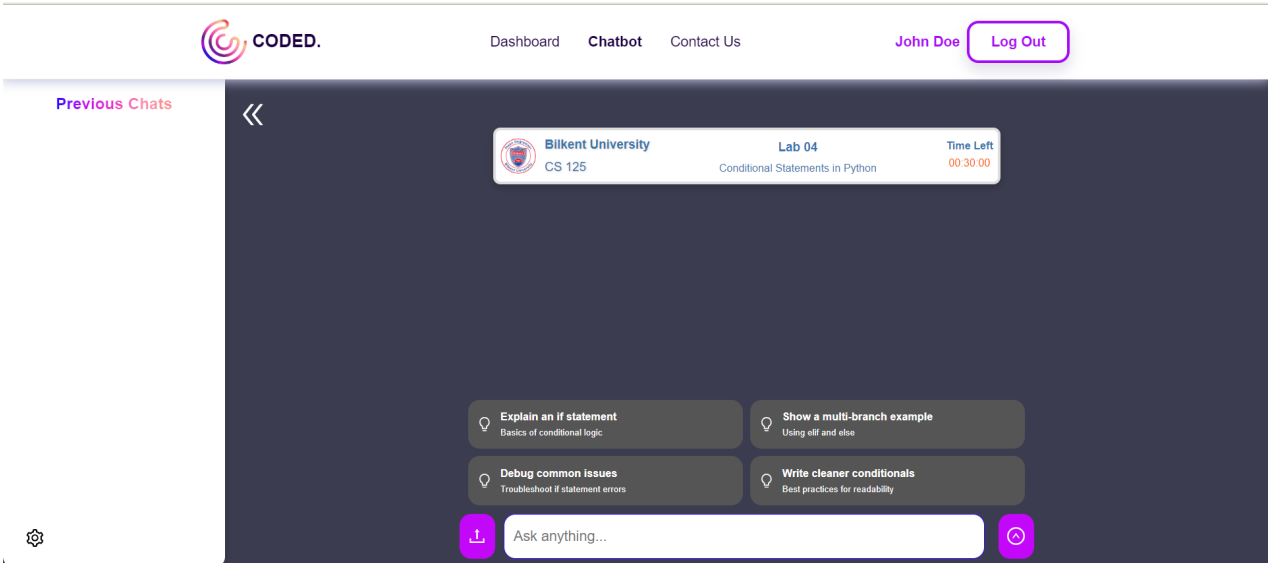

Fig. 3.5.5.2.9: Chatbot Page

In this page, students can directly ask their questions to the chatbot. Students can see their previous chats with the bot, which is located inside the sliding bar, shown left in the image. Also, related CS/CTIS lab's information together with university's information and the time left for this specific lab are shown. Students may choose pre-generated questions, which are generated by chatbot itself. Students might upload file which has code extension (such as .py) to the chatbot to evaluate, or they can write/copy paste their questions to the input box.

3.5.5.3. Instructor Interfaces

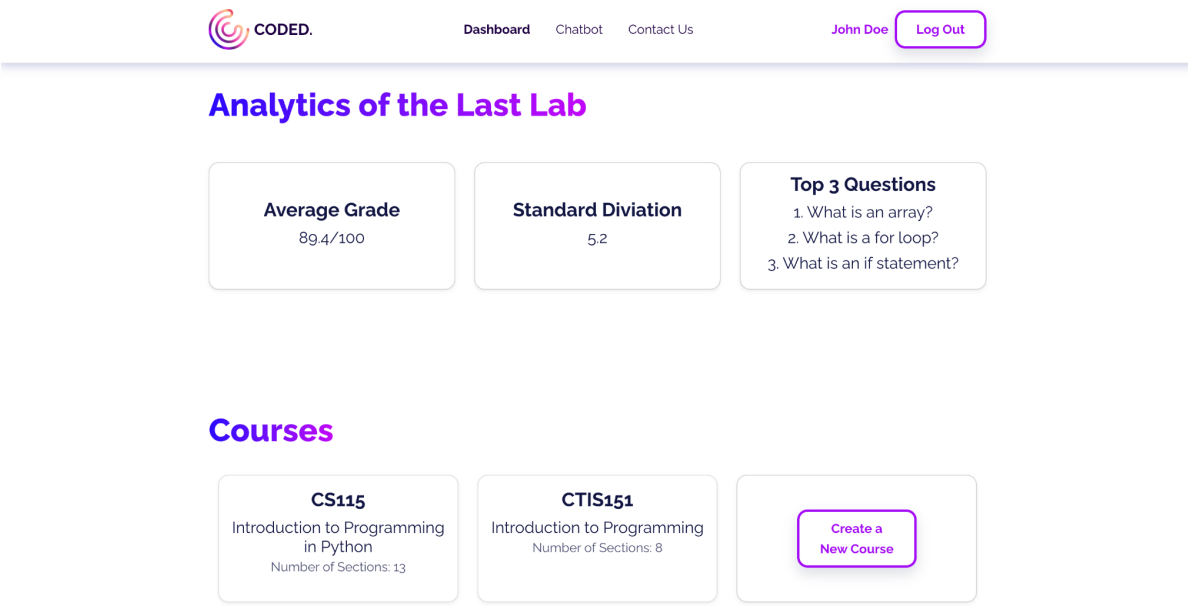

Fig. 3.5.5.3.1: Instructor Dashboard

The dashboard page for the instructors are designed so that the instructors can view the essential information as soon as they login to the website. Therefore, the analytics from the last lab of one of their courses are displayed at the top of the page. If they need to check other course material, they can select their wanted course from the ‘Courses’ menu. If they would like to add another course to *CODED.,* they can do so by pressing the ‘Create a New Course’ button, which will redirect them to the ‘Add Course’ page.

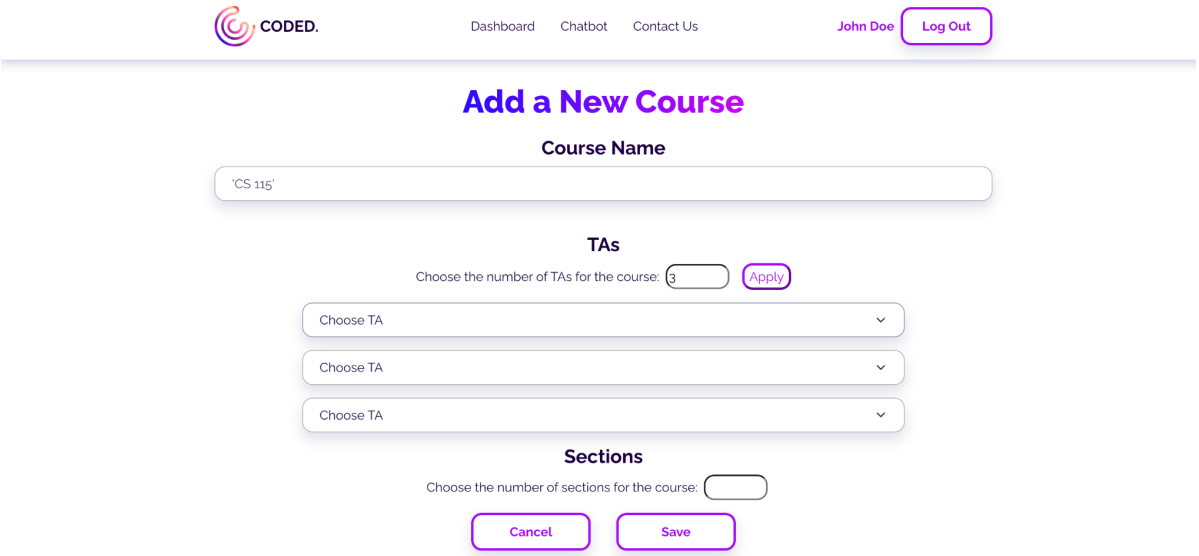

Fig. 3.5.5.3.2: Add a New Course Page

The add course page is used to add another course to the system. Only the instructors are allowed to create a new course. If they want to create another course, they have to specify the course name, the TAs that will be responsible for the course, and the number of sections for the course. After all of the necessary fields are filled, the course will be reachable through the dashboard.

Lab Upload Page for Similarity Checks

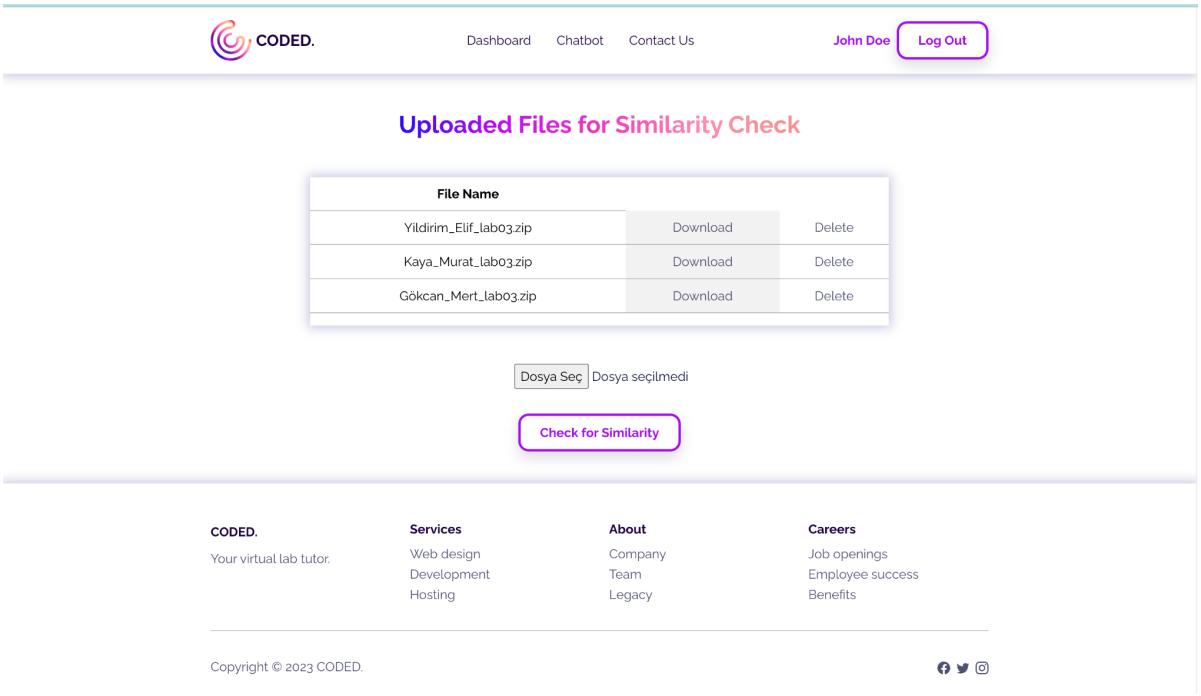

Fig. 3.5.5.3.3: Lab Upload Page for Similarity Check

This page is only accessible by people who have grading access, and in here, the lab assignment of each student can be uploaded for a similarity check. News labs can be added, and existing labs can be deleted for the similarity check.

Similarity Check Results Page

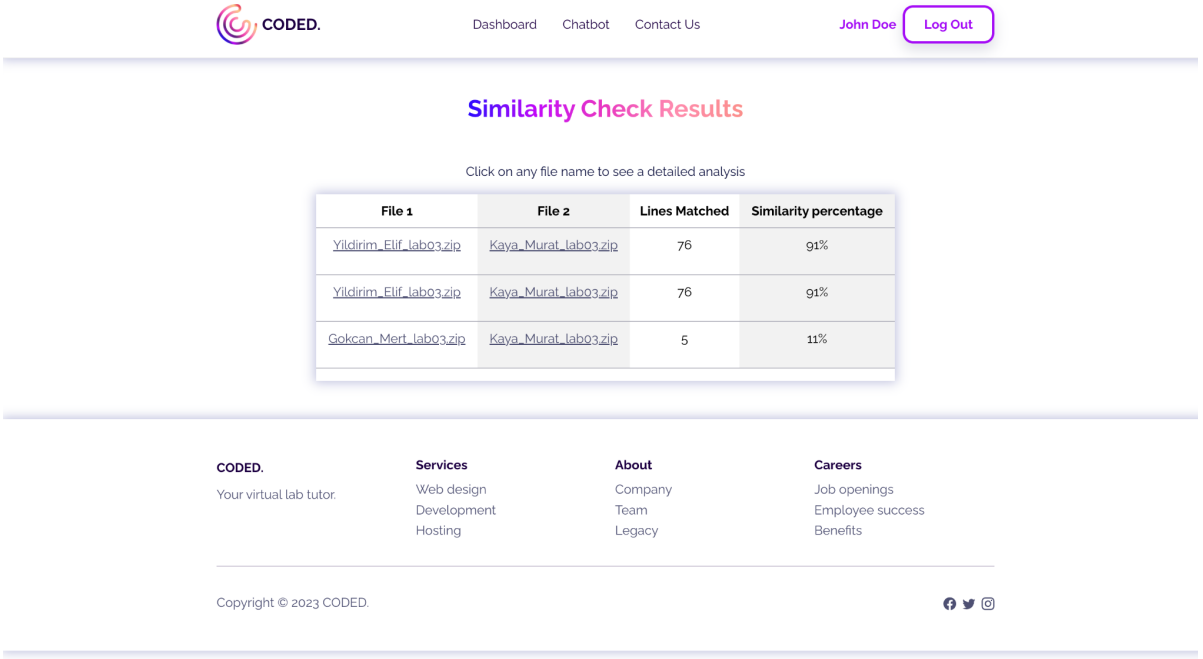

Fig. 3.5.5.3.4: Similarity Check Results Page

This page is accessible only to individuals with grading access. It provides detailed information on the similarity percentages for each uploaded lab. The program conducts comparisons between each lab and other submissions, performing similarity checks across all possible pairs. It identifies the number of matched lines and calculates the similarity percentage for each pair. Instructors, TAs, and tutors can easily identify submissions with higher plagiarism probabilities. They have the option to further analyze these labs by clicking on the respective lab names.

4. Other Analysis Elements

4.1. Consideration of Various Factors in Engineering Design

As a part of engineering practice, how the ***CODED.*** system is constructed regarding factors such as Data Privacy, Accessibility & Equality, Public Welfare, Global, Cultural, Social, Environmental, and Economic constraints are discussed in detail.

4.1.1. Data Privacy Considerations

CODED. considers Data Privacy to be a highly regarded issue. As one of the main features, *CODED.*'s chatbot integrates sophisticated measures to ensure compliance with the **General Data Protection Regulation (GDPR)** and strictly uses **Amazon Relational Database Service (RDS)** for secure and efficient data management. The chatbot's design, which is a fine-tuned version of Meta AI's ***Code Llama*** model, follows GDPR's requirements for personal data protection, ensuring that user data is processed lawfully, transparently, and for legitimate purposes only [4]. Chatbot and its cloud server, which is deployed on the ***Amazon Sagemaker***, utilize prosperous encryption and anonymization techniques to safeguard sensitive user information [5].

Moreover, the utilization of Amazon RDS in *CODED.'s* infrastructure plays a crucial role in maintaining data privacy. Amazon RDS provides a highly secure environment for database management, featuring encryption at rest and in data transmitting, automated backups, and network isolation mechanisms [6]. This aligns with GDPR's mandates for employing advanced security measures to prevent data breaches and unauthorized access.

In addition to technical safeguards, *CODED.*'s chatbot is designed with a user-centric approach to data privacy. It transparently communicates its data handling practices, seeks

explicit user consent for data collection, and provides users with control over their personal data. This includes options for data access, rectification, and erasure, aligning with GDPR's principles of user rights and data minimization.

4.1.2. Accessibility & Equality Considerations

One of the main important factors for the engineering design of *CODED.* is making the CS and CTIS lab process accessible to everyone in an equal manner. This commitment to **equality** is evident in both the overall application design and the specific features like the chatbot. The application has been developed with a user-friendly interface that accommodates diverse user needs, including those who may not be proficient in technology, considering the fact that the target people of *CODED.* are the ones who learn programming as beginners. This includes clear navigation, readable fonts, and a responsive design that is compatible with various devices and screen sizes, ensuring **accessibility** for all users.

The chatbot, a prominent feature of *CODED.*, is designed following ethical guidelines to avoid biases in interactions. This approach ensures that the chatbot interactions are fair and unbiased, providing equal support to all users regardless of their background [7]. The *CODED.*, which uses Code Llama algorithms as a base model, is developed to avoid discriminatory responses and provide consistent, reliable information to all users, reinforcing the equality principle in educational resources [8].

Additionally, the overall application and the chatbot are regularly updated to reflect the latest educational trends and user feedback, ensuring that the system evolves to meet the changing needs of a diverse user base. This adaptive approach demonstrates *CODED.*'s commitment to providing an equitable learning environment for all students and faculty members in CS and CTIS labs.

4.1.3. Public Welfare Considerations

By facilitating a better learning environment, *CODED.* contributes to skill development among students. This is crucial for preparing a skilled workforce, which is a critical component of public welfare, as it directly impacts employability and economic growth [9].

Besides employability and economic growth, *CODED.* emphasizes a new trend in computer science: large language models (LLMs) [10]. According to initial market research, it is concluded that AI in education is emerging, and CODED. is hereby the key pioneer in this area, accounting for its wide range of features that have never been done by any startup before. It is an indicator of *CODED.'s* public welfare impact in the area of education, favoring Artificial Intelligence.

4.1.4. Global Considerations

CODED. is precisely designed to be adaptable for a wide range of universities worldwide, particularly those following the US or European education systems. It aims to be a globally-oriented product, ensuring its performance is not impeded by varying academic structures and requirements. The application's design is highly customizable, allowing educators to tailor features to their specific needs, enabling immediate integration into their computer science labs.

Recognizing the prevalence of English as a medium of instruction in many universities, especially in the United States, *CODED.* primarily uses English. This choice supports its global applicability. However, to further extend its global reach, *CODED.* is also available in Turkish, Spanish, and French upon request. Future updates aim to include additional languages and localization features, catering to a broader international audience and addressing cultural nuances more effectively.

Enhancements in *CODED.* also focus on supporting diverse curricula, accommodating different grading systems, and aligning with various academic calendars. Moreover, AI components like chatbots are developed with cultural sensitivity in mind, ensuring appropriateness and inclusivity in interactions [11]. Accessibility features are also a priority, adhering to international standards to serve a broader range of users, including those with disabilities [12].

To ensure *CODED.* meets the diverse needs of its global user base, ongoing collaborations with international educational institutions are established for continuous feedback and development. This approach helps *CODED.* stay responsive to changing educational trends and regulations across different regions. Additionally, the platform is optimized for varied network environments, ensuring efficient performance worldwide, and commits to sustainable, eco-friendly practices in its development and deployment.

4.1.5. Cultural Considerations

One of the prior goals for *CODED.* is to pay attention to the cultural or other aspects of differences between users, favoring inclusiveness and diversity in the environment. The content, including examples, scenarios, and interactions within the platform, especially in the chatbot, would need to be culturally sensitive. This involves avoiding stereotypes, using inclusive language, and being mindful of cultural nuances and differences.

As an education application, *CODED.* should also consider the fact that cultural differences may influence learning styles [13]. *CODED.*'s design and teaching methodologies should accommodate various learning preferences and educational backgrounds, ensuring that all students can benefit equally from the resources.

In terms of data privacy and/or other regulations that matter for the *CODED.*, it should be complementary with the cultural background and the influence of that culture on the regulations to obey terms and regulations within the country/state.

4.1.6. Social Considerations

CODED. capacity to effect positive social change, particularly in computer science education, is substantial. Efforts could be intensified to reach not only underprivileged groups but also those who are typically underrepresented in STEM fields, such as women and minorities. This commitment to diversity and inclusion would significantly contribute to leveling the educational playing field and promoting social equity.

In addition to broadening access, *CODED.* is instrumental in identifying and supporting students who face challenges in lab sessions. The platform could be enhanced with sophisticated analytics tools, enabling instructors and teaching assistants to gain deeper insights into individual student performance. These tools can provide detailed, actionable data, allowing for timely and personalized intervention.

Furthermore, engaging students in feedback and development processes can empower them and provide valuable insights for the continual improvement of *CODED*. This inclusive approach ensures that the platform evolves to meet the diverse needs and preferences of its user base, thus enhancing its social impact and relevance in the ever-changing landscape of computer science education.

4.1.7. Environmental Considerations

CODED.'s transition of education to a digital platform plays a pivotal role in reducing the environmental footprint of CS and CTIS labs in a limited perception. By digitizing many traditional paper-based processes, especially in areas like grading and assignment submission, *CODED.* significantly diminishes paper usage, contributing to a reduction in deforestation and waste.

In addition to minimizing paper waste, *CODED.*'s compatibility with remote learning setups presents an opportunity for further environmental benefits. Facilitating online

education potentially reduces the need for students and academic staff to commute, thereby lowering carbon emissions associated with transportation[14].

However, an area of environmental concern in *CODED.* is the use of the chatbot model ***CodeLlama***, which, despite its advanced capabilities, has a considerable carbon footprint due to high computing costs. The reliance on intensive GPU usage for this chatbot inevitably leads to increased electricity consumption and, consequently, higher carbon emissions [15]. While this is a current limitation in balancing the project's environmental goals with its technical needs, it is acknowledged as an area for future improvement.

To address this challenge and align more closely with sustainable practices, *CODED.* could explore alternative, more energy-efficient AI models or invest in offsetting its carbon footprint through renewable energy usage or other sustainability initiatives. Moreover, engaging with ongoing research in reducing the environmental impact of chatbots and AI systems is crucial. As technology evolves, there is optimism that more sustainable solutions will emerge, allowing *CODED.* to maintain its technological edge while improving its environmental sustainability.

4.1.8. Economic Considerations

The economic framework of *CODED.* encompasses several key aspects. Firstly, efficient budget management is crucial to ensure that funds are judiciously allocated for development, maintenance, and updates. This involves a thorough cost-benefit analysis to justify the investment by weighing the platform's benefits against its operational expenses.

Resource optimization, particularly in human resource deployment, is essential to enhance productivity while minimizing costs. Exploring various revenue models, such as subscriptions or educational grants, can provide sustainable financial support for *CODED*.

Moreover, maintaining economic accessibility is vital, making *CODED.* affordable or potentially free for educational institutions, especially those with constrained budgets, like the universities of the least developed countries. This approach not only broadens the platform's reach but also reinforces its commitment to educational equity. Also, the long-term economic impact of *CODED.* should be considered, particularly its role in enhancing skills and employability among students, thereby contributing positively to broader economic development.

4.1.9. Table of the Aforementioned Considerations

Consideration	Effect Degree [0 - 10]	Comment
Data Privacy	10	*The most important factor.*
Accessibility & Equality	8	*One of the most important ones.*
Public Welfare	4	*Less important.*
Global	8	*Being a global key software is very important for CODED.*
Cultural	6	*Somewhat important.*
Social	6	*Somewhat important.*
Environmental	3	*Due to limitations, it has very little effect for now.*
Economic	7	*As a non-invested startup, the economic situation is important for CODED.*

Table 1: Considerations of Engineering Factors Summary Table

4.2. Risks and Alternatives

In this section, the possible risks and alternative ways (B plans) to handle these risks will be discussed.

4.2.1. Time Management

While *CODED.*'s project plan is carefully structured with clear deadlines, unforeseen challenges in achieving these timelines may arise. The team's prior experience working together and ongoing communication significantly reduces this risk. Each team member's familiarity with others' working styles and strengths contributes to a cohesive work environment, further minimizing potential delays.

However, **in the occasion that time management issues do arise**, a proactive and structured approach will be implemented to address them. This includes but is not limited to:

1. **Regular Progress Reviews**: Implementing more frequent progress reviews together with the team and supervisor, Prof. Güvenir, to identify and address any delays early in the process.
2. **Resource Reallocation**: If certain tasks are lagging, resources can be reallocated to focus on these areas, ensuring that the project remains on track.
3. **Schedule Adjustments**: Revising the project schedule where necessary, allowing for a more realistic and achievable timeline while maintaining the quality and scope of the project.
4. **Risk Buffering**: Incorporating buffers into the project timeline to accommodate unforeseen delays, thus reducing pressure on the team and maintaining project momentum.

5. **Enhanced Communication and Reporting**: Strengthening communication channels within the team to ensure all members are updated on progress and challenges, facilitating a collaborative approach to problem-solving.

By adopting these strategies, the project can effectively navigate any time management challenges that arise, ensuring that *CODED.* progresses smoothly toward its completion.

4.2.2. Generative AI (GenAI)

For the chatbot implementation in the *CODED.* project, the team selected the **Code LLama model with 7B parameters**, an open-source option, during the initial meetings. As of this report's drafting, the *CODED.* team has successfully run the model and obtained preliminary results on Google Colab Pro with V100 extended GPU server. Following the presentation of these results to the **client** (Bilkent University Department of CTIS) and **project supervisor** (Prof. Güvenir), the decision was made to deploy this model on an **AWS** server to enhance response time efficiency.

However, this decision entails certain risks and necessitates the consideration of alternative strategies. The primary risks include:

- **Dependency on External Services**: Relying on AWS exposes the project to risks associated with external service providers, such as potential downtime or changes in service policies.
- **Cost Overruns**: AWS services, while scalable, can lead to unforeseen costs, especially if the usage exceeds the estimated projections.
- **Technical Compatibility and Performance Issues**: The integration of **Code LLama** with AWS might encounter technical challenges that could affect performance.

To prevent these risks, the following alternatives and strategies could be considered:

- **Redundancy Plans**: Establish backup systems or secondary service providers (such as Google Cloud and Microsoft Azure) to ensure continuity in case of AWS service disruptions.
- **Cost Management Strategies**: Implement robust monitoring and alerts for AWS usage to control costs and prevent budget overruns. The team will also contact representatives of AWS or other service provider executives to get free credits to use the *CODED*. system.
- **Technical Readiness and Testing**: Conduct thorough, complex, and robust testing of the **Code LLama** model on AWS to identify and resolve compatibility or performance issues before full-scale deployment.

4.2.3. Work Distribution

The project engages five undergraduate students from Computer Science, providing a team sufficiently sized to meet the project's requirements. Despite this, scheduling conflicts may arise due to the significant individual workloads of these students, potentially hindering their ability to consistently collaborate. Given the necessity for cohesive assembly of the project components, dividing the work into separate, independent sections is not deemed an efficient approach.

To navigate this challenge, the project will adopt a step-by-step execution model. This will involve organizing brief, regular meetings focused on discussing and planning subsequent phases of the project, ensuring continuity and coordination among team members.

4.2.4. Summary of Risks and Alternatives

Potential Risks	Likelihood	Effect	B Plan
Time Management	***Medium*** ↔	***Medium*** ↔	Replan the work schedule to prevent any further problems.
Generative AI	***Medium*** ↔	***High*** ↑	Adopt OpenAI API, which is deployed on the GPT server itself.
Work Distribution	***Low*** ↓	***Low*** ↓	Redistribute work distribution by taking into account their expert areas.

Table 2: Summary of Risks and Alternatives

4.3. Project Plan

WP#	Work Package Title	Leader	Members Involved
WP 1	Project Specification Report	Zeynep Hanife Akgül	Beyza Çağlar Selin Bahar Gündoğar Emre Karataş Zeynep Selcen Öztunç
WP 2	Analysis and Requirement Report	Zeynep Selcen Öztunç	Zeynep Hanife Akgül Beyza Çağlar Selin Bahar Gündoğar Emre Karataş
WP 3	Frontend Development	Beyza Çağlar	Zeynep Hanife Akgül

			Selin Bahar Gündoğar Emre Karataş Zeynep Selcen Öztunç
WP 4	Backend Development	Emre Karataş	Zeynep Hanife Akgül Beyza Çağlar Selin Bahar Gündoğar Zeynep Selcen Öztunç
WP 5	Initial Demo	Zeynep Hanife Akgül	Beyza Çağlar Selin Bahar Gündoğar Emre Karataş Zeynep Selcen Öztunç
WP 6	Backend Development of Features	Zeynep Selcen Öztunç	Zeynep Hanife Akgül Beyza Çağlar Selin Bahar Gündoğar Emre Karataş
WP 7	Detailed Design Report	Beyza Çağlar	Zeynep Hanife Akgül Selin Bahar Gündoğar Emre Karataş Zeynep Selcen Öztunç
WP 8	Beta Release	Selin Bahar Gündoğar	Zeynep Hanife Akgül Beyza Çağlar Emre Karataş Zeynep Selcen Öztunç
WP 9	Project Launch	Emre Karataş	Zeynep Hanife Akgül Beyza Çağlar Selin Bahar Gündoğar Zeynep Selcen Öztunç

Table 3: *List of work packages*

WP 1: Project Specification Report			
Start Date: 01.10.2023 **End Date:** 03.11.2023			
Leader:	Zeynep Hanife Akgül	**Members Involved:**	Beyza Çağlar Emre Karataş Selin Bahar Gündoğar Zeynep Selcen Öztunç
Objectives: The goal of this report is to give a detailed explanation of CODED. by specifying requirements and making feasibility analysis.			
Tasks: **Task 1.1** Creation of High-Level System Architecture & Components of Proposed Solution Diagram **Task 1.2** Deciding and Explaining Constraints **Task 1.3** Deciding and Explaining Requirements **Task 1.3.1** Non-Functional Requirements **Task 1.3.2** Functional Requirements **Task 1.4 Feasibility Discussions:** Making feasibility analysis on every feature of CODED. **Task 1.5 References:** Ensure that citations are listed and properly formatted for all utilized sources in the report in accordance with the relevant citation guidelines.			
Deliverables **D1.1** Project Specification Document			

Table 4: *Work Package 1*

WP 2: Analysis and Requirements Report			
Start Date: 18.10.2023 **End Date:** 08.12.2023			
Leader:	Zeynep Selcen Öztunç	**Members Involved:**	Beyza Çağlar Emre Karataş Selin Bahar Gündoğar Zeynep Hanife Akgül

Objectives: The goal of this report is to define CODED.'s features and requirements, and to present key system models including use-case, sequence, state, and activity diagrams.
Tasks: **Task 2.1** Scenarios **Task 2.2** Creation of Use-Case Diagram **Task 2.3** Creation of Object and Class Model **Task 2.4 Creation of Dynamic Models:** Create activity, sequence, and state diagrams. **Task 2.5** Creation of UI Designs **Task 2.6 Other Analysis Elements:** Examine critical considerations for CODED., including data privacy, accessibility, and environmental impact, assess risks and alternatives, and discuss ethical responsibilities. Outline a project plan and strategies for team learning and technical knowledge acquisition. **Task 2.7 References:** Ensure that citations are listed and properly formatted for all utilized sources in the report in accordance with the relevant citation guidelines.
Deliverables **D2.1** Analysis and Requirements Report

Table 5: *Work Package 2*

WP 3: Frontend Development			
Start Date: 15.10.2023 **End Date:** 15.12.2023			
Leader:	Beyza Çağlar	**Members Involved:**	Zeynep Hanife Akgül Selin Bahar Gündoğar Emre Karataş Zeynep Selcen Öztunç
Objectives: Designing the display mock-ups and implementing the interface while maximizing the user experience for all user types.			
Tasks: **Task 3.1** User Experience Research			

Task 3.2 Mock-Up Design **Task 3.3 Technology Research and Decision:** Analyze different technologies and decide on the one that offers the best user experience and the most optimal performance for *CODED*. **Task 3.4** Initialization of the React Project **Task 3.5** Implementation of the Interface based on Mock-Up Designs
Deliverables **D3.1** User Interface Design Mock-Ups **D3.2** User Interface on the Web

Table 6: *Work Packag*e 3

WP 4: Backend Development			
Start Date: 22.10.2023 **End Date:** March 2024			
Leader:	Emre Karataş	**Members Involved:**	Zeynep Hanife Akgül Beyza Çağlar Selin Bahar Gündoğar Zeynep Selcen Öztunç
Objectives: Implementation of the backend of the application according to the design proposed in the Analysis and Requirements Report.			
Tasks: **Task 4.1** Initializing SpringBoot Project with Correct Dependencies **Task 4.2** Implementation of Classes According to Class Diagram given in Section 5.1.2. **Task 4.3** Mapping Chatbot Server with JPA Interface **Task 4.4** Service Layer Implementation of *CODED*. **Task 4.5** Connecting External Features' Backend Implementation (explained in WP6) with the Service Layer **Task 4.6** Implementation of Database on AWS RDS Server, located in Osaka, Japan. **Task 4.7** Implementing Controller Classes **Task 4.8:** Testing Controller Endpoints via Postman Tool **Task 4.9:** Connecting the Backend with Frontend			

Task 4.10: Dockerizing the Backend Application **Task 4.11:** Deployment to AWS by Dockerizing the App
Deliverables **D4.1** The Backend Application **D4.2** Docker Container

Table 7: *Work Package 4*

WP 5: Initial Demo			
Date: 15.12.2023			
Leader:	Zeynep Hanife Akgül	**Members Involved:**	Beyza Çağlar Selin Bahar Gündoğar Emre Karataş Zeynep Selcen Öztunç
Objectives: Presentation of the project so far, by deploying it through an example scenario.			
Tasks: **Task 5.1 Preparation of the Presentation:** Prepare a slide presentation to give an overview of the project. **Task 5.1.1 Presentation Work Division:** Assign each member to a part of the slide to present. **Task 5.2 Preparation of Demo Flow:** Decide on a demo flow of which interfaces to follow on the demo, by creating an imaginary scenario of a whole lab process. **Task 5.2.1 Demo Work Division:** Assign each member to interfaces to explain the functionalities. **Task 5.2 Rehearsal:** Do rehearsals of the oral presentation and be prepared for the possible questions from the audience and the juries.			
Deliverables **D5.1:** Oral Demo Presentation			

Table 8: *Work Package 5*

WP 6: Backend Development of the Features			
Start Date: 22.10.2023 **End Date:** March 2024			
Leader:	Zeynep Selcen Öztunç	**Members Involved:**	Zeynep Hanife Akgül Beyza Çağlar Selin Bahar Gündoğar Emre Karataş
Objectives:			
Tasks: **Task 6.1:** Research Similarity Check APIs **Task 6.2:** Integrate MOSS API into the Project **Task 6.3:** Test the Similarity Check Feature **Task 6.4:** Fine Tuning base Code LLama model **Task 6.5 Deployment of Llama:** Deploy the fine-tuned version of Code LLama model to AWS and using its API to interact with basic frontend modules. **Task 6.5:** Research Test Case Runner APIs **Task 6.6:** Integrate pytest and UnityTest into the Project **Task 6.7:** Test the Test Case Runner Feature **Task 6.8:** Complete the SonarQube API implementation for the code analyzer property			
Deliverables **D6.1:** Fully implemented backend application containing a similarity checker, chatbot, code analyzer and test case runner			

Table 9: *Work Package 6*

<table>
<tr><td colspan="4">WP 7: Detailed Design Report</td></tr>
<tr><td colspan="4">Start Date: January 2024 End Date: February 2024</td></tr>
<tr><td>Leader:</td><td>Beyza Çağlar</td><td>Members Involved:</td><td>Zeynep Hanife Akgül
Selin Bahar Gündoğar
Emre Karataş
Zeynep Selcen Öztunç</td></tr>
<tr><td colspan="4">Objectives: The objective of this report is to analyze the technical details, methodologies, and decisions made in the design of the project to provide better understanding.</td></tr>
<tr><td colspan="4">Tasks:
Task 7.1 Proposed Software Architecture:
Task 7.1.1 Subsystem Decomposition: Break down the system into smaller, more manageable parts to make it more organized and understandable.
Task 7.1.2 Hardware/Software Mapping: Determine how the software components are matched to run optimally on the hardware components, while ensuring efficient utilization of resources and overall system performance.
Task 7.1.3 Persistent Data Management: Organize and maintain data, ensuring its durability, accessibility, and integrity within the application.
Task 7.1.4 Access Control and Security: Implement systems that prevent una
Task 7.2 Subsystem Services: Explain how the functionalities provided by individual subsystems contribute to the overall functionality of the application.
Task 7.3 Test Cases
Task 7.4 Engineering Design Considerations: Ensure that the project meets engineering constraints and standards.
Task 7.5 Teamwork Details: Explain the work distribution among the team members.
Task 7.6: Glossary and References</td></tr>
<tr><td colspan="4">Deliverables
D7.1 Detailed Design Report</td></tr>
</table>

Table 10: *Work Package 7*

WP 8: Beta Release			
Start Date: April 2024 **End Date:** May 2024			
Leader:	Selin Bahar Gündoğar	**Members Involved:**	Beyza Çağlar Emre Karataş Zeynep Selcen Öztunç Zeynep Hanife Akgül
Objectives: The goal of the beta release is to test the application for its performance, efficiency, customer satisfaction, and discovering bugs and issues before releasing it to a wide-range audience.			
Tasks: **Task 8.1** Final Product Meeting **Task 8.2** Completion of the Backend Development **Task 8.3** Completion of the Frontend Development **Task 8.4** Deploying the Website on the Server **Task 8.5 Lab Observation:** Arrange dates with the CTIS151, CTIS152, CS115, and CS125 instructors to observe students using CODED. **Task 8.6 Testing *CODED.*:** Prepare student, instructor, TA/tutor satisfactory survey to test customer satisfaction and receive feedback **Task 8.7 Feedback Loop:** Every feedback will be used to improve the application until the next lab session			
Deliverables **D8.1** Satisfaction Survey			

Table 11: *Work Package 8*

WP 9: Project Launch			
Date: May 2024			
Leader:	Emre Karataş	**Members Involved:**	Beyza Çağlar Zeynep Selcen Öztunç Selin Bahar Gündoğar Zeynep Hanife Akgül
Objectives: Successfully deploy the web application for user access.			
Tasks: **Task 9.1 Deploy to Web Server:** Set up and configure the web server for hosting the application, to make sure it's accessible to users on various web browsers. **Task 9.2 Optimize for Web Browsers:** Fine-tune the web app to ensure compatibility and optimal performance across different web browsers like Chrome, Firefox, Safari, and Edge. **Task 9.3 Establish Secure Connections:** Implement and configure SSL certificates to secure the web application, providing safe and encrypted user connections.			
Deliverables **D9.1** Web Application **D9.2** Final Report & Final Demo			

Table 12: *Work Package 9*

4.4. Ensuring Proper Teamwork

Since our project consists of many components that require hard work and detailed analysis, ensuring proper teamwork is a key concept in order for us to finish this project on time. We realized this earlier in the project, and that is why we have been using Jira from the very start of the project. Jira is a leading agile project management tool widely used by teams for planning, tracking, releasing, and supporting high-quality software [16]. We use its timeline feature to add our deadlines, such as reports and meetings, in order to help us

visualize how much time we have left for various tasks. We also use its Kanban board feature, which is an agile project management tool that enhances work visualization, limits ongoing work, and boosts efficiency, benefiting teams in organizing and completing their tasks effectively [17]. We have columns on this board such as “To-do,” “In Progress,” “Finished,” “Being Tested,” and “Done,” which represent the different steps of our tasks. Each team member is responsible for updating the progress of their tasks on the Kanban board.

There are also other software that we use to ensure proper teamwork other than Jira. We are using WhatsApp and Zoom for communication, GitHub for version control, Google Drive for working on reports and documents, keeping logs, and Lucidchart and diagrams.net software to work on diagrams.

Other than using online software for teamwork, we also meet up 2 or 3 times a week to keep each other informed about our progress and the challenges that we are facing. We also keep logs of our meetings. It is also worth mentioning that we divide the workload evenly so that no member is doing significantly larger tasks than others. We try to ensure proper teamwork by doing all the steps mentioned above.

4.5. Ethics and Professional Responsibilities

CODED. is designed to guide students through their labs while relieving the responsibilities of tutors and instructions. While this eases the lab experience of students, tutors, and instructors, this application can also raise critical ethical issues due to its comprehensive nature, and we are committed to addressing them. All of the interactions with the chatbot will be confidential since we utilize user privacy. The application will adhere strictly to local Turkish data protection laws and the internationally recognized General Data Protection Regulation (GDPR) [18].

The chatbot is designed to handle various inputs, including inappropriate ones. Instead of ignoring such questions, the chatbot will guide the student to ask related questions to the lab or code. The grading that is done by the system also has well-defined parameters, and it analyzes the logic of individual methods. This ensures that students can earn partial credit for their efforts. However, the final authority on grading remains with the course instructors. For transparency, students will have access to detailed feedback on how their submissions were evaluated, ensuring they understand areas of strength and improvement. Access to the system will be through unique course-specific entrance codes to protect student information. This ensures that only the users who enrolled in the specific course can access the course content. The application can also detect similar code patterns, which might indicate plagiarism. However, the final decision on whether the code is plagiarized is left to the instructors. So, the system only provides insights on whether the code is plagiarized, and instructors can decide on the matter based on the context.

4.6. Planning for New Knowledge and Learning Strategies

Throughout the project, we will use some systems and technologies that we have little or no familiarity with. All of us are familiar with Spring Boot and React framework, but some of us have no familiarity with software such as UnityTest, pytest, Hugging Face, SonarQube, and MOSS. We realized this earlier in our project and have already started using/testing these software in order to understand how they work so that we can integrate the APIs of these software into our project. Getting hands-on experience with this software greatly helped us to learn about them. When in need, we will watch online courses and will go through the documentation of this software. We will also do an in-depth analysis of APIs and web crawlers in general.

For our chatbot feature, we will use ***Code-Llama***, and most of us have no experience using it. To effectively learn and utilize ***Code Llama***, we will utilize all of the relevant online material, and we will actively experiment with various prompts for code generation and debugging to integrate ***Code Llama*** into our project. Also, during the learning phase, more experienced team members will provide assistance and guidance to those with less experience. Additionally, we will balance the time invested in learning new technologies with our project deadlines, ensuring that this essential skill development does not disrupt our overall timeline.

5. Glossary

- **Amazon Relational Database Service (RDS):** A web service that simplifies the setup, operation, and scaling of a relational database for use in applications. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing users to focus on their applications and business. Amazon RDS supports several types of database engines, including MySQL, PostgreSQL, MariaDB, Oracle, and Microsoft SQL Server [19].
    - **Acronym: RDS**
- **Artificial Intelligence (AI)**: An acronym that refers to the simulation of human intelligence in machines that are programmed to think and act like humans. This includes capabilities such as learning, reasoning, self-correction, and problem-solving.[19].
    - **Acronym**: AI
- **Application Programming Interface (API)**: A set of protocols and tools that allow different software applications to communicate with each other, enabling the exchange of data and functionalities [19].
    - **Acronym**: API
    - **Example**: The Hugging Face API allows developers to access and utilize a variety of machine learning models for natural language processing tasks, such as text generation, translation, and sentiment analysis.
- **Computer Science (CS)**: The study of computers and computational systems, focusing on algorithms, software design, and the theoretical underpinnings of computing [19].
    - **Acronym**: CS

- **Comma-Separated Values (CSV)**: A file format used to store tabular data, such as a spreadsheet or database, in plain text. Each line in a CSV file corresponds to a row in the table, and each field in that row (or cell in the table) is separated by a comma. This format is widely supported by many applications and is useful for transferring data between different programs [19].
    - **Acronym:** CSV
- **General Data Protection Regulation(GDPR)**: It is a regulation in EU law on data protection and privacy in the European Union and the European Economic Area [19].
    - **Acronym**: GDPR
- **Information Systems and Technologies (CTIS)**: An interdisciplinary field that combines computing technology with business practices to facilitate data storage, analysis, and communication within organizations [19].
    - **Acronym**: CTIS
- **Large Language Models (LLM)**: Machine learning models trained on extensive datasets to understand and generate human-like text based on the input they receive [19].
    - **Acronym**: LLM
    - **Example**: GPT-4 is an example of a large language model used for various natural language processing tasks.
- **Measure of Software Similarity (MOSS)**: An automated system for detecting plagiarism in software assignments. It is commonly used in academic settings to compare student submissions and identify similarities that may indicate copying [19].
    - **Acronym**: MOSS

    - **Example**: Many universities use MOSS to maintain academic integrity in computer science courses by checking if students have plagiarized code for their assignments.
- **Science, Technology, Engineering, and Mathematics (STEM)**: An interdisciplinary approach to learning that integrates these four disciplines into a cohesive curriculum focused on real-world applications [19].
    - **Acronym**: STEM
    - **Example**: Many educational initiatives aim to boost student interest and competence in STEM fields due to their importance in the modern workforce.
- **Teaching Assistant (TA)**: An individual, often a graduate or undergraduate student, assists a professor in instructional responsibilities, including grading, answering student queries, and occasionally teaching [19].
    - **Acronym**: TA
- **Two Factor Authentication(2FA)**: A security process that requires users to provide two different forms of identification before gaining access to an account or system. This adds an extra layer of security compared to single-factor authentication [19].
    - **Acronym**: 2FA

6. References

[1] K. Emre, "Discussion on CS labs in Bilkent University," Academic interview, Oct. 28, 2023.

[2] "Code smell," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Code_smell. [Accessed: Oct. 28, 2023].

[3] S. Pegarella, "Privacy Policy for Chatbots," TermsFeed, 01 July 2023. [Online]. Available: https://www.termsfeed.com/blog/privacy-policy-chatbots/. [Accessed: Dec. 07, 2023].

[4] M. Hasal et al., "Chatbots: Security, privacy, data protection, and social aspects," Concurrency Computat. Pract. Exper., vol. 33, no. e6426, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/cpe.6426. [Accessed: Nov. 28, 2023].

[5] "Amazon Web Services, Inc., 'Data Protection in Amazon SageMaker - Amazon SageMaker,' Amazon SageMaker Documentation. [Online]. Available: https://docs.aws.amazon.com/sagemaker/latest/dg/data-protection.html. [Accessed: Nov. 28, 2023].

[6] Amazon Web Services, Inc., 'Data protection in Amazon RDS - Amazon Relational Database Service,' Amazon Relational Database Service User Guide. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/DataDurability.html. [Accessed: Nov. 28, 2023].

[7] J. Bang, S. Kim, J. W. Nam, and D.-G. Yang, "Ethical Chatbot Design for Reducing Negative Effects of Biased Data and Unethical Conversations," in Proc. 2021 Int.

Conf. Platform Technol. Service (PlatCon), 2021. DOI: 10.1109/PlatCon53246.2021.9680760. [Accessed: Nov. 28, 2023].

[8] L. Ranaldi, E. S. Ruzzetti, D. Venditti, D. Onorati, and F. M. Zanzotto, "A Trip Towards Fairness: Bias and De-Biasing in Large Language Models," 2023. [Online]. Available: https://arxiv.org/pdf/2305.13862.pdf. [Accessed: Nov. 28, 2023].

[9] T. Zobel, H. Steinbeck, and C. Meinel, "Towards Inclusive Education: A Review of and Guide to Accessible Features in Chatbots for MOOCs," 2023 IEEE Learning with MOOCS (LWMOOCS), 2023, pp. 1-5. DOI: 10.1109/LWMOOCS58322.2023.10306062.

[10] L. A. Jacques, "Teaching CS-101 at the Dawn of ChatGPT," acm Inroads, vol. 14, no. 2, pp. [pp 3-6], June 2023. DOI: 10.1145/3595634.

[11] C. Kelsey, "AI Chatbot to Increase Cultural Relevancy of STEM," Indiana University News, Indiana University, Oct. 17, 2023. [Online]. Available: https://news.iu.edu/live/news/31938-ai-chatbot-to-increase-cultural-relevancy-of-stem. [Accessed: Dec. 4, 2023].

[12] S. Federici, M. L. de Filippis, M. L. Mele, S. Borsci, M. Bracalenti, G. Gaudino, A. Cocco, M. Amendola, and E. Simonetti, "Inside Pandora's Box: A Systematic Review of the Assessment of the Perceived Quality of Chatbots for People with Disabilities or Special Needs," Disability and Rehabilitation-Assistive Technology, vol. 15, no. 7, pp. 832-837, 2020. DOI: 10.1080/17483107.2020.1775313.

[13] V. Prabhakaran, R. Qadri, and B. Hutchinson, "Cultural Competencies in Artificial Intelligence," presented at the First Workshop on Cultures in AI/AI in Culture (non-archival), NeurIPS 2022. [Online]. Available:

[https://ai-cultures.github.io/papers/Cultural_Competencies_in_Artificial_Intelligence__NeurIPS_2022_Culture_AI_Workshop.pdf. [Accessed: Dec. 4, 2023].

[14] M. P. Jarillo, L. Pedraza, P. M. Ger, and E. Bocos, "Challenges of Online Higher Education in the Face of the Sustainability Objectives of the United Nations: Carbon Footprint, Accessibility and Social Inclusion," Sustainability, vol. 11, no. 20, p. 5580, 2019. DOI: 10.3390/su11205580.

[15] "Introducing Code Llama, a state-of-the-art large language model for coding," ai.meta.com, August 24, 2023. [Online]. Available: https://ai.meta.com/blog/code-llama-large-language-model-coding/ [Accessed: Dec. 4, 2023].

[16] Atlassian, "What is Jira Software?," Atlassian.com, [Online]. Available: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software. [Accessed: Nov. 24, 2023].

[17] Atlassian, "What is a Kanban board," Atlassian.com, [Online]. Available: https://www.atlassian.com/agile/kanban/boards. [Accessed: Nov. 24, 2023].

[18] The European Parliament and the Council of the European Union, “General Data Protection Regulation,” 27-Sep-2022. [Online]. Available: https://gdpr-info.eu/. [Accessed: Oct. 31, 2023].

[19] OpenAI, "ChatGPT [Large language model]," 2023. [Online]. Available: https://chat.openai.com.