Software Requirements Specifications for PeerSet

Prepared By Christina Constas and Alexandre Meyer

Version 5.7 Approved 12/10/21
Last Revised 12/10/21

Table of Contents

Revision History

Name	**Date**	**Changes**	**Version**
Christina Constas & Alexandre Meyer	9/1/21	Created the document and started defining the purpose and scope before stakeholder meetings.	**1.0**
Christina Constas & Alexandre Meyer	9/6/21	Added Table of Contents, added to Introduction, and fixed some formatting issues.	**1.1**
Alexandre Meyer & Christina Constas	9/7/21	Continued adding to the Introduction and fixing formatting issues Added to the Overview of the document, User Characteristics, Constraints/Design Constraints	**1.2**
Christina Constas	9/8/21	Continued editing Sections 1 & 2 Added "TBD" to empty sections	**1.3**
Christina Constas & Alexandre Meyer	9/9/21	Continued editing Section 1 Continued formatting and adding to Section 2 (System Interfaces, User Interfaces, Hardware Interfaces) Added to portability, product functions, Security, Assumptions, Logical Database Requirements and Design Constraints	**1.4**
Christina Constas	9/10/21	Continued adding to Section 2 (User Interfaces)	**1.5**
Christina Constas &	9/13/21	Started adding to External	**1.6**

Alexandre Meyer		Interfaces, added more information to Performance requirements	
Christina Constas & Alexandre Meyer	9/14/21	Added to Section 1 (Scope), Section 2 (User Interfaces, Product Functions) and Section 3 (Software System Attributes)	**1.7**
Christina Constas & Alexandre Meyer	9/15/21	Added to Section 2 (User Interfaces)	**1.8**
Christina Constas & Alexandre Meyer	9/16/21	Added to Section 2 (User Interfaces, Operations, Memory Constraints)	**1.9**
Christina Constas & Alexandre Meyer	9/17/21	Added to Section 2 (User Interfaces) and Section 3 (External Interfaces)	**2.0**
Christina Constas & Alexandre Meyer	9/22/21	Added to Section 2.2 (Product Functions)	**2.1**
Christina Constas & Alexandre Meyer	9/23/21	Edited Sections 1, 2 (System Interfaces, Product Functions) and 3 (Functions, Design Constraints)	**2.2**
Christina Constas & Alexandre Meyer	9/24/21	Edited Sections 2.1.2 (User Interfaces), 2.2 (Product Functions) and 3.2 (Functions)	**2.3**
Alexandre Meyer	9/25/21	Edited Table of Contents	**2.4**
Christina Constas & Alexandre Meyer	9/27/21	Edited Section 2.2 (Product Functions) and Table of Contents	**2.5**
Christina Constas & Alexandre Meyer	9/28/21	Vocabulary updates Edits to Overall Description and Section 2.1.2 (User Interfaces)	**2.6**

Christina Constas & Alexandre Meyer	9/29/21	Vocabulary updates Edits to Sections 2.1.2 (User Interfaces), 2.1.3 (Hardware Interfaces), 2.1.4 (Software Interfaces) and 2.1.6 (Operations)	**2.7**
Alexandre Meyer	9/30/21	Edits to Sections 2.4 (Constraints) and 3.5 (Design Constraints) to reflect the use of ReactJS for front end development	**2.8**
Christina Constas & Alexandre Meyer	10/1/21	Edits to Section 3.2 (Functions)	**2.9**
Christina Constas & Alexandre Meyer	10/5/21	Edits to Section 3.4 (Logical Database Requirements)	**3.0**
Christina Constas & Alexandre Meyer	10/6/21	Edits to Sections 1.2 (Scope), 2.1.2 (User Interfaces) and 2.1.6 (Operations) Edited Sections 2.1.3 (Hardware Interfaces) and 2.5 (Assumptions & Dependencies) to reflect stakeholder requirement that the software is incompatible with mobile devices	**3.1**
Christina Constas & Alexandre Meyer	10/7/21	Updated Section 1.3 to include administrator role definition Updated Sections 2.4 and 3.5 to reflect the use of mySQL database Updates to Section 3.1 (External Interfaces)	**3.2**

Christina Constas & Alexandre Meyer	10/8/21	Edited Section 2.2 (Product Functions) to reflect UML diagram from Engine	**3.3**
Christina Constas & Alexandre Meyer	10/11/21	Edited Sections 2.2 (Product Functions) and 3.2 (Functions) to reflect UML diagram from Engine	**3.4**
Christina Constas & Alexandre Meyer	10/12/21	Edited Section 2.2 (Product Functions) to reflect UML diagram from Engine Updated Section 3.1 (External Interfaces)	**3.5**
Christina Constas & Alexandre Meyer	10/13/21	Edited Section 3.1 (External Interfaces)	**3.6**
Christina Constas & Alexandre Meyer	10/14/21	Edited Section 3.1 (External Interfaces), replaced all instances of placeholder name with “PeerSet”	**3.7**
Christina Constas & Alexandre Meyer	10/15/21	Edited Section 3.1 (External Interfaces)	**3.8**
Christina Constas & Alexandre Meyer	10/18/21	Edited Section 3.1 (External Interfaces)	**3.9**
Christina Constas & Alexandre Meyer	10/19/21	Edited Section 3.1 (External Interfaces)	**4.0**
Christina Constas & Alexandre Meyer	10/26/21	Edited Sections 2.1.3 (Hardware Interfaces) and 3.4 (Logical Database Requirements	**4.1**
Christina Constas & Alexandre Meyer	10/27/21	Edited Section 3.4 (Logical Database Requirements)	**4.2**

Christina Constas & Alexandre Meyer	10/28/21	Updated Section 3.8 (Other Diagrams) with with professor and student user flow diagrams from GUI Changed all instances of the word "instructor" to "professor" Edited Sections 1.3 (Definitions, Acronyms & Abbreviations) and 2.1.1 (System Interfaces)	**4.3**
Christina Constas & Alexandre Meyer	10/29/21	Edited Sections 2.1.1 (System Interfaces), 2.3 (User Characteristics), 2.4 (Constraints), 3.5 (Design Constraints), 3.6.2 (Availability), 3.6.3 (Security) and 3.6.5 (Portability)	**4.4**
Christina Constas & Alexandre Meyer	11/2/21	Edited Sections 2.1.1 (System Interfaces) and 3.4 (Logical Database Requirements)	**4.5**
Christina Constas & Alexandre Meyer	11/3/21	Edited Sections 2.1.1 (System Interfaces) and 2.1.2 (User Interfaces)	**4.6**
Christina Constas & Alexandre Meyer	11/5/21	Edited Sections 2.1.2 (User Interfaces) and 3.1 (External Interfaces)	**4.7**
Christina Constas & Alexandre Meyer	11/9/21	Made formatting changes and minor copy edits Edited Sections 2.1.2 (User Interfaces) and 3.4 (Logical Database Requirements)	**4.8**

Christina Constas & Alexandre Meyer	11/12/21	Edited Sections 2.2 (Product Functions) and 3.2 (Functions)	**4.9**
Christina Constas & Alexandre Meyer	11/15/21	Edited Sections 2.2 (Product Functions) and 3.2 (Functions) Added UML diagram from Team Engine and MIT License to Section 4 (Supporting Information)	**5.0**
Christina Constas & Alexandre Meyer	11/16/21	Edited Sections 2.1.2 (User Interfaces), 3.1 (External Interfaces), 3.2 (Functions) and 3.4 (Logical Database Requirements	**5.1**
Christina Constas & Alexandre Meyer	11/17/21	Edited Section 2.1.2 (User Interfaces)	**5.2**
Christina Constas & Alexandre Meyer	11/19/21	Edited Sections 2.1.2 (User Interfaces) and 3.1 (External Interfaces)	**5.3**
Christina Constas & Alexandre Meyer	11/29/21	Edited Sections 2.1.1 (System Interfaces), 2.1.2 (User Interfaces), 3.1 (External Interfaces), and 3.2 (Functions)	**5.4**
Christina Constas	12/3/21	Edited Section 2.1.2 (User Interfaces)	**5.5**
Christina Constas & Alexandre Meyer	12/4/21	Edited Sections 2.1.1 (System Interfaces), 2.1.2 (User Interfaces), 2.2 (Product Functions), and 3.1 (External Interfaces)	**5.6**
Christina Constas & Alexandre Meyer	12/10/21	Edited Sections 2.1.1 (System Interfaces), 2.1.2 (User Interfaces),	**5.7**

		3.1 (External Interfaces), and 3.2 (Functions)	

1. Introduction

1.1. Purpose

The purpose of this document is to delineate the system requirements for PeerSet, a web-based application that will be used to facilitate calibrated peer review (CPR) in higher education courses. CPR is a method of peer review in which students are provided with calibrated examples of "poor," intermediate," and "high" quality solutions to assignments set by their professors. These serve as a baseline for their evaluations of other students' work (Tenbergen & Daun, 2022).

The audience of this document includes the design and development teams that will be working to build PeerSet. Product stakeholders may be viewing this document as well. Stakeholders include professors, who will ultimately be using PeerSet to facilitate CPR in their courses, and designers and developers at IBM, who have required the design and development teams to use specific tools to build PeerSet.

1.2. Scope

PeerSet will be a web-based application used by both students and professors to conduct CPRs. PeerSet must be able to do the following:

1. Distribute assignments, and collect student or team solutions to assignments at the end of a designated preparation period
2. Distribute review assignments to students or teams. No student or team should be reviewing their own work, each solution should receive the same number of reviews, and each student or team must have the same number of review assignments (or review assignments must be distributed as evenly as possible with the number of teams or students enrolled; for example, one team may review 4 solutions in one week, then 3 solutions the next week)
3. Distribute review packages to students or teams. Packages include anonymized student solutions, instructions, calibrated examples of "poor," "intermediate," and "high" quality solutions, and an professor-prepared template for documenting grades and feedback
4. Collect student feedback and grades

5. Rename submitted files and collate feedback for manual quality checks by professors
6. Distribute feedback to students or teams

In addition to the functional requirements listed above, PeerSet must be built using IBM's Open Liberty software development framework. It must utilize a microservices architecture, which must be developed using Eclipse MicroProfile. JSON Web Tokens (JWTs) must be used for security and authentication.

The primary goal of PeerSet is to make the process of CPR facilitation easier and less time-consuming for professors. With an improved CPR process, professors are better positioned to help students reap the benefits of CPR, which include increased exposure to both adequate and inadequate solutions. This exposure makes students aware of alternative solutions to problems, and helps them learn to identify mistakes to avoid in the future. Additionally, studies have shown that peer review can improve student grade outcomes by up to 14%, and learning outcomes (as measured by midterm and final exam grades) showed improvement as well (Tenbergen & Daun, 2022).

1.3. Definitions, Acronyms & Abbreviations

- **Administrator:** Any user who is able to install and run the system, or elevate the privileges of other users
- **Assignment:** Distributed to students by professors. An assignment poses a problem to which the students must prepare a solution.
- **CPR:** Calibrated Peer Review.
- **GUI:** Graphical User Interface. In a microservices architecture, the GUI maintains the state of the application and facilitates communication between the user and the microservices.
- **Professor:** Any user who is instructing a course.
- **JWTs:** JSON Web Tokens. Used authenticate users and secure requests between services.
- **Microservices Architecture:** A software architecture in which each component of the program is an independent, fully-functioning application, or microservice.
- **REST:** REpresentational State Transfer. Is used to perform create, read, update and delete (CRUD) functions within resources.

- **Review Assignment:** The anonymized student solutions that students or teams are assigned to review.
- **Review Package:** Materials that are provided to students for conducting reviews.
- **Solution:** The answer provided by students to the problem posed in an assignment. Solutions are submitted to the professor at the end of a designated preparation period.
- **Submission:** Any document that has been uploaded to the system.
- **Student:** Any user who is enrolled in a course as a student.

1.4. References

- Introduction to Calibrated Peer Review
- Activity: Manually Managing CPR
- Calibrated Peer Reviews in Requirements Engineering Instruction: Application and Experiences (Tenbergen & Daun, 2022)
- OpenLiberty
- JSON Web Tokens
- Eclipse MicroProfile
- Microprofile-starter - This is a microprofile tutorial to get familiar with
- CISQ Software Reliability Standards (for future reference)
- REST APIs (IBM Cloud Education)
- U.S. Federal Code of Regulation 34 CFR Part 99

1.5. Overview

Section 2 of this document contains the overall description of PeerSet, while Section 3 contains specific requirements. Section 3 will be used to implement PeerSet while ensuring that the requirements outlined in Section 2 will be met. Section 4 contains supporting information and includes an index and appendixes.

2. Overall Description

PeerSet is a web application that will be used by students and professors in the facilitation and completion of CPR. The professors must be able to create and manage one course or multiple courses (including multiple sections of the same course). Additionally, students must be able to enroll in one course or multiple courses. To start, both students and professors must log into the application, using their university emails and passwords. Students will be reviewing a certain number of other students' submissions anonymously. This number will be specified by the professor and dependent upon the number of students who are enrolled in the course. If students are completing and reviewing solutions as teams, this number will be dependent on the number of teams in the course. Once the solutions have been reviewed, they must be resubmitted.

2.1. Product Perspective

PeerSet is self-contained and independent of any larger system.

2.1.1. System Interfaces

PeerSet will utilize a microservices architecture, in which the GUI will facilitate communication between the user and the microservices. The following subsections describe the APIs within each of the microservices and the functions they fulfill within the system.

2.1.1.1. Login Microservice

2.1.1.1.1. Login

This API manages logins. It will handle the authentication of users.

2.1.1.1.2. User

This API will manage user information. It will be used to create a user, get the information of all users, get a specific user, update a user's information, and delete a user.

2.1.1.2. Professor Microservice

2.1.1.2.1. Professor

This API will manage professor information. It primarily handles student and assignment information associated with the professor, as well as quality assessments and rejections of assignments.

2.1.1.3. Student Microservice

2.1.1.3.1. Student

This API will manage student information. It will be used to create a student from a user, get the information of all students, get a specific student, update a student's information, and delete a student.

2.1.1.4. Utility Microservice

2.1.1.4.1. Assignment

This API will manage assignments. It will be used to create assignments, get all assignments, get a specific assignment, and delete an assignment.

2.1.1.4.2. Course

This API will manage courses. It will be used to create a course, get a specific course, and delete a specific course.

2.1.1.4.3. Review

This microservice will manage reviews. It will be used to store reviews, get all reviews, get a specific review, update a review, and delete a review.

2.1.1.4.4. Submission

This API will manage submissions. It will be used to store submissions, get all submissions, get a specific submission, update a submission, and delete a submission.

2.1.1.4.5. CSV Import

This API will use data from CSV files uploaded by professors to create student users and add them to courses.

2.1.2. User Interfaces

The following subsections describe each of the necessary user interfaces for interaction with PeerSet.

2.1.2.1. All Users

The following subsections describe interfaces for supporting functions that must be performed by all users of PeerSet.

2.1.2.1.1. Login/Logout

Both students and professors will need to log in to use PeerSet. Students and professors must log in using their university Google credentials. Colleges and universities must have the ability to whitelist their domains, so only users with email addresses from that domain will be able to log in.

To start, users must use the buttons on the login page to select their role (either student or professor). From there, they will be redirected to the Google login page. After they enter the correct information, users will be logged in to Peerset. To log out of Peerset, users must click on their username, located in the upper right corner of each page. Clicking the username will cause a drop-down button to appear, giving the user the option to sign out of PeerSet.

2.1.2.1.2. Help Documentation

A user manual will be provided with the software, detailing the steps for both student and professor interactions with PeerSet. A deployment manual will also be provided with the software, detailing the steps for installation and other deployment-related activities.

2.1.2.2. Students

Students will be interacting with PeerSet in three roles: as solution submitters, as reviewers, and as viewers of feedback. The interfaces described in the following subsections must be designed to support each role.

2.1.2.2.1. Main Navigation

The main navigation must be visible on all screens of the application. For students, the main navigation will include three tabs: Assignments (which will direct students to the Assignments by Course page), Results (which will direct students to the Results page), and Teams (which will direct students to the Teams page). Additionally, the student's username will be displayed on the right side of the navigation; clicking the username will give students the option to sign out of PeerSet. A home icon to the left of the username will direct students back to the Home Dashboard. The PeerSet logo, which will be located on the left side of the navigation bar, will also direct students to the Home Dashboard.

2.1.2.2.2. Student Home Dashboard

The student dashboard will be the first screen that students see after login. Students must be able to easily access upcoming assignments, information about their teams, and recent review results from this page.

2.1.2.2.3. Assignments By Course

Students must be able to access both upcoming and past assignments from the Assignments by Course page. Assignments must be organized by due dates, with the most recent deadlines being displayed near the top of the screen. Tabbed navigation will allow students to move between assignment lists for each of their courses. Assignments may be in one of two states: upcoming (yet to be submitted) and completed (with solutions submitted). Students must be able to filter their assignment lists, so only assignments in a specific state will be displayed. By default, the system will display all assignments. Students will be able to access their team information from the Assignments by Course page by clicking the “View Teams” button in the upper right of the page.

2.1.2.2.4. Solution Assignment Pages

Students must be able to view and download assignment instructions, in the form of PDF documents uploaded by professors. Students must also be able to submit solutions through these pages at the end of each designated preparation period, and the system must provide appropriate feedback so students are certain that their solutions are submitted successfully. The system must be able to check submissions and alert students if there are any issues with their submission that need to be addressed, such as a name. Students must have the opportunity to resubmit solutions until the submission deadline has passed. If students are working in teams, each team member must be able to sign off on each submission.

2.1.2.2.5. Peer Review Assignment Pages

Students must be able to easily access their review packages once professors make them available. Elements of the review package such as templates and calibrated solution examples will be the same for all students, but assigned submissions for review must be unique to each student or team.

Students must be able to submit reviews of their peers' solutions through the application, and the system must provide appropriate feedback so students are assured that their reviews were successfully submitted.

2.1.2.2.6. Results

Students must be able to view and download feedback from their peers once it is available. Students must receive email notifications from the system when feedback becomes available for viewing.

2.1.2.3. Professors

The following subsections describe interfaces that must be designed to support professors in their role as facilitators of CPR.

2.1.2.3.1. Main Navigation

The main navigation must be visible on all screens of the application. For professors, the main navigation will include three tabs: Courses & Assignments (which will direct professors to the Courses & Assignments page), Quality Check (which will direct professors to the Quality Check page), and Students & Teams (which will direct professors to the Students & Teams page). Additionally, the professor's username will be displayed on the right side of the navigation; clicking the username will give professors the option to sign out of PeerSet. A home icon to the left of the username will direct professors back to the Home Dashboard. The PeerSet logo, which will be located on the left side of the navigation, will also direct professors to the Home Dashboard.

2.1.2.3.2. Professor Home Dashboard

The professor home dashboard will be the first screen that professors see after they log in. From the dashboard, professors must be able to view their existing courses, create a new course, and access an assignment portal, where they will be able to create new assignments and view existing assignments and submissions. Professors must also be able to access peer review results from the professor dashboard.

2.1.2.3.3. Courses & Assignments

From the Courses & Assignments page, professors must be able to create new courses and assignments. They must also be able to view each of the assignments for their existing courses. Assignments will be displayed as lists, with the due dates for solutions and peer reviews displayed together underneath the assignment title. When an assignment moves from the solution phase to the peer review phase, the system will mark the solution as completed. Assignment lists will be organized by due date; assignments with the nearest due dates will be at the top of the list. Completed assignments must be placed at the bottom of the assignment list. Tabbed navigation will allow professors to move between assignment lists for each of their courses. Assignments may be in one of two states: draft (or unpublished) and active (published and ongoing). Professors must be able to filter their assignment lists, so only assignments in a specific state will be displayed. By default, the system will display all assignments.

2.1.2.3.4. Creating a Course

When creating a new course, professors must be able to add students by uploading a CSV file containing student I.D. numbers and email addresses. There must also be a field where professors can manually enter a course title; it is recommended that the title includes a course reference number. Professors must be able to add more than one course; additionally, they must be able to add multiple sections of the same course. Professors

When creating a course, professors will have the ability to either make teams of individual students or randomized with a set number of teams.

2.1.2.3.5. Build New Assignment

When creating assignments, professors must have the ability to upload assignment sheets for both solutions and peer reviews as PDF files. The interface must also have a field for naming the assignment; the system will add, “Solution,” or, “Peer Review,” to the assignment name as the students progress through each of the phases. For each assignment, professors must be able to choose if students will be working in randomized teams, or as individuals. If the professor selects “randomized teams,” then they must enter the number of teams they wish to create, so the system can distribute students as evenly as possible. Professors must be able to enter deadlines for both solution and review submissions. Each submission will be timestamped, so professors will know if students have submitted work after the specified deadline. Once an assignment moves from the solution phase to the peer review phase, the window for solution submissions will be closed.

2.1.2.3.6. Viewing and Editing Assignments

From the Courses & Assignments page, professors will be able to access the pages for individual solution and review assignments. Each page title will match the title of the assignment. Each assignment page will display the date the assignment was published, the solution deadline and the peer review deadline. Assignment pages will also be tagged as either individual or team assignments. Professors will be able to edit assignment pages by clicking an "Edit Assignment" button in the upper right corner of the page. A PDF display will allow professors to view the instructions for each phase of the assignment, and tabs will allow them to move between the solution instructions and the peer review instructions.

2.1.2.3.7. Students & Teams

On the Students & Teams page, professors will be able to view a list of all students enrolled in each of their courses. A toggle button will allow them to move back and forth between the student list and the teams list. Teams may be expanded, so professors can view the students on each team. Professors will be able to manually add or delete students from the Students & Teams page, as well.

2.1.2.3.8. Solution Quality Check

The system cannot completely automate quality assessments of student work, but professors must be able to conduct these quality assessments with the assistance of the system. The system must be able to scan submitted solutions for any elements that might undermine the CPR process, such as student names or profanity. Once the scan has been completed, a View Errors dropdown shall appear. Clicking the dropdown will display a list of the errors that were discovered within the solution. This will allow the professor to assess whether or not an element is inappropriate in the context of the assignment. Once the professor has assessed the provided error list, they will be presented with two options: they will be able to either fix the errors themselves and resubmit the solution, or they will be able to reject the solution, which will notify the student that their solution has errors that must be fixed. The professor can select teams from a sidebar to view the selected team solutions.

2.1.2.3.9. Distributing Review Assignments

Professors must have an interface that facilitates efficient, quasi-random distribution of review assignments to students. Review assignments must be distributed as evenly as possible for the number of submissions and students or teams in each class. Additionally, the system must ensure that no student or team is assigned to review their own work. Each submission must receive the same amount of reviews, and each team must review the same amount of submissions.

If the number of students or teams doesn't allow for an even distribution of review assignments, some students or teams may need to review extra submissions. If this is the case, the professor must be able to ensure that the extra reviews are distributed evenly across review periods, so no team is reviewing extra submissions for several review periods in a row. Professors must be able to quickly view the number of reviews assigned to each student or team. They must also be able to quickly view which teams were assigned to review which submissions. If the professor doesn't approve of the distribution of review assignments, they must be able to quickly change the distribution before making review assignments available to students.

2.1.2.3.10. Peer Review Quality Check

As with the solution quality check, the system must be able to scan student reviews for elements such as names or profanity that might undermine the CPR process. The system will display each of these instances in an error list, allowing the professor to more easily assess reviews and determine whether or not to accept or reject student PDFs.

Professors must be able to quickly and easily view the grades that were given and received by each student or team, including average grades for each student or team. This will allow them to quickly scan for outlier grades that need to be addressed.

The professor can select teams from a sidebar to view the selected team Reviews. Once the professor has selected the team submissions they need to check, buttons for the individual reviews will direct them to the specific review for that team. Once professors have finished the quality check, they will be able to distribute the reviews to students by clicking the "Submit Results" results button in the top right corner of the page.

2.1.3. Hardware Interfaces

PeerSet will be a desktop, web-based application. Therefore, it must be compatible for use with desktop and laptop computers running any operating system and any web browser. PeerSet must not be compatible with mobile devices (such as tablets or smartphones), and the system must actively discourage users from trying to log in using those devices.

2.1.4. Software Interfaces

PeerSet must be developed as a RESTful API using:

- JAX-RS: Will be used in the development of the RESTful API web services.
- Maven: Will be used as a project management tool.
- JSON-B: Will be used to store JSON in binary format.
- Open Liberty: Is a framework developed by IBM that will be used to run the servers for the application and the database.
- JWTs: These will be tokens used to authenticate users.
- Eclipse MicroProfile: Will be used in the development of the microservice architecture.

2.1.5. Memory Constraints

This web application must not have more than 150-200MB of ram\primary memory usage. Currently, there is no limit to the amount of secondary memory usage.

2.1.6. Operations

All Users:

- Can log into the system
- Can view help documentation/tutorials specific to their roles

Administrator

- Users with administrator privileges can install and run the system

Professors:

- Can create courses
  - Can add students to courses (either by CSV upload or manually)
  - Can remove students from courses.
- Can create teams.

    - Can change teams at any time or for specific assignments.
- Can create assignments.
    - Can close submissions after the deadline.
- Can distribute review assignments.
- Can conduct quality assessments with assistance from the system.
- Can make feedback available to students.
- Can see student grades for each assignment, including averages.
- Can receive student feedback about reviewed assignments.

Students:
- Can upload assignment solutions.
- Can upload assignment reviews.
- Can view feedback on their own assignment solutions from other students.

2.2. Product Functions

LoginAPI
- **getSpecificUser():** This function will authenticate a user. If the authentication fails, the user will receive a 404 error. Otherwise, a JWT for the user will be generated.

UserAPI
- **getAllUsers():** This function will display all users in the system for an administrator
- **getSpecificUser():** This function will display a specific user in the system
- **postUser():** This function will add a new user to the database
- **updateUser():** This function will update a user in the database
- **deleteSpecificUser():** This function will delete a user from the database. Only administrators should be able to delete all users; professors should be able to delete students.

ProfessorAPI
- **getStudentsAndTeams():** This will get the Students and teams for a course.

- **getAssignmentsByProfessor():** This function will get all assignments. associated with a specific professor, based on the professor's userID.
- **getQualityCheckReviewsByProfessor():** This function will get reviews for the Quality Check, based on the assignmentID.
- **getQualityCheckSolutionsByProfessor():** This function will get solutions for the Quality Check, based on the assignmentID.
- **rejectSolution():** This function will send an email notifying students if their solution is rejected by their professor.
- **rejectReview():** This function will send an email notifying students if their review is rejected by their professor.
- **assignReviewForTeams():** This function will randomly assign reviews to teams as evenly as possible for the enrollment number. Students will be notified that they have a new review assignment.
- **setResultStage():** This function will set an assignment from solution to review state.

CSVImport

- **parse():** This function will parse CSV files for information to create users.
- **makeTeamsAutomatically():** This function will randomly assign students to teams based on parameters set by the professor.
- **addTeamsToDatabase():** This function will update student teamIDs in the database.
- **removeSpecialCharacters():** This function will remove special characters from the csv data.
- **addToDatabase():** This function will add user information from the CSV to the database.
- **makeStudentWithoutSpecialCharacters():** This function will will remove special characters from all student information.

SubmissionAPI

- **getAllSubmissions():** This function will display students' submitted solutions.
- **getSpecificSubmission():** This function will display a submitted solution from a specific student or team.

- **postSubmission():** This function will be used to submit student solutions.
- **updateSubmission():** This function will replace the existing submission if the student re-submits a solution.
- **deleteSpecificSubmission():** This function will delete a student submission. Only professors should be able to delete submissions.

ReviewAPI

- **getAllReviews():** This function will display students' submitted reviews.
- **getSpecificReview():** This function will display a submitted review from a specific student or team.
- **postReview():** This function will be used to submit student reviews.
- **updateReview():** This function will replace the existing submission if the student re-submits a review.
- **deleteSpecificReview():** This function will delete a student review. Only professors should be able to delete reviews.

AssignmentAPI

- **getAllAssignments():** This function will display all assignments in a list format for the user.
- **getSpecificAssignment():** This function will display a specific assignment description if the user clicks on that assignment.
- **postAssignment():** This function will create an assignment object using professor input.
- **updateAssignment():** This function will edit an assignment object using professor input.
- **deleteSpecificAssignment():** This function will remove an assignment object from the database. Only professors should be able to delete assignments.

StudentAPI

- **getAllStudents():** This function will display all of the students enrolled in a specific course.
- **getSpecificStudent():** This function will display a single student enrolled in a course.

- **postStudent():** This function will create a student user.
- **updateStudent():** This function will update a student user.
- **deleteSpecificStudent():** This function will remove a student user from the database. Only administrators and professors should be able to delete students.

CourseAPI

- **getAllCourses():** This function will display a list of each of the user's courses.
- **getSpecificCourse():** Uses a course ID to retrieve data from a specific course from the database.
- **postCourse():** This function will create a course object using professor input.
- **updateCourse():** This function will edit a course object based on professor input.
- **deleteSpecificCourse():** This function will remove a course object from the database. Only administrators and professors should be able to delete courses.

2.3. User Characteristics

The intended users of this application are college students at the undergraduate level, and their professors. Users with any sort of technical expertise should be able to use this application.

2.4. Constraints

PeerSet must allow for an infinite number of courses and assignments. Users must be authenticated before use of this application. Professors must be able to activate courses with a CSV file of student names and ID numbers; they must also be able to add or delete students manually. Students may be enrolled in multiple courses, but may only view courses in which they are enrolled. Assignments, grades, rubrics, and feedback must be housed. Each team's assigned reviews must be randomly generated by the application. A n number of reviews will be assigned to each team (excluding self review). Solutions and Reviews must be submitted through the system, and students must have the ability to resubmit solutions or reviews until the submission window is closed. All user data will be stored in a mySQL database. REACTJS will be used in the development of the user interface of this application.

2.5. Assumptions & Dependencies

This application is being developed on the assumption that professors and students using their university emails will be using this application. The application is also being developed on the assumption that the system will only be accessed via desktop or laptop computers, and will never be accessed with a mobile device.

3. Specific Requirements

This section of the SRS document will contain software requirements that shall help with the development, design and testing of the system.

3.1. External Interfaces

The following subsections detail system outputs in response to potential user inputs.

3.1.1. All Users

This subsection details possible interactions between the system and all users.

3.1.1.1. Login Page

Input: User selects their role
Output: User is directed to the Google login screen
Input: User correctly enters their university Google credentials
Output: User is redirected to Home Dashboard
Input: User incorrectly enters their university Google credentials
Output: User is notified that their credentials are invalid and is prompted to to reenter
Input: User attempts to access the system using a smartphone or tablet
Output: The system displays a message explaining that it is incompatible with mobile devices and that the user must log in using a computer

3.1.2. Student Interfaces

This subsection details possible interactions between the system and users in the student role.

3.1.2.1. Main Navigation

Input: Student clicks “Home” tab

Output: Student is directed to the Home Dashboard

Input: Student clicks “Assignments” tab

Output: Student is directed to the Assignments by Course page

Input: Student clicks “Teams” tab

Output: Student is directed to the Teams page

Input: Student clicks “Results” tab

Output: Student is directed to the Results page

Input: Student clicks on the PeerSet logo

Output: Student is directed to the Home Dashboard

3.1.2.2. Home Dashboard (Student)

Input: Student clicks the “See All” button next to Upcoming Assignments

Output: Student is directed to Assignments by Course page.

Input: Student clicks an item under Upcoming Assignments

Output: Students are directed to the assignment page for the clicked item

Input: Student clicks on Manage Teams

Output: Student is directed to the Teams page

Input: Student clicks on a course under Recent Results

Output: Student is directed to the results for that course on the Results page

Input: Student clicks on an individual review result under Recent Results

Output: Student is directed to the Peer Review Results page for the clicked result

3.1.2.3. Assignments by Course (Student)

Input: Student clicks a course tab

Output: Dashboard displays a task list for that course

Input: Student clicks an item on the task list for the displayed course

Output: Student is directed to the page for the clicked item

Input: Student clicks “Upcoming” radio button

Output: Dashboard displays only assignments that have yet to be submitted
Input: Student clicks “Completed” radio button
Output: Dashboard displays only completed assignments
Input: Student clicks “All” radio button
Output: Dashboard displays all assignments

3.1.2.4. Submitting Solutions

Input: Student clicks “Download Instructions” button
Output: Assignment instructions are downloaded to the student’s computer
Input: Student clicks the right arrow of the PDF display
Output: System displays the next page of the assignment instructions
Input: Student clicks the left arrow of the PDF display
Output: System displays the previous page of the assignment instructions
Input: Student clicks “Upload” button
Output: The student’s file manager is displayed, allowing them to select a document from their hard drive
Input: Student selects a PDF to submit
Output: PDF is attached
Input: Student clicks “Submit”
Output: Solution is submitted; system sends an email to each member of the team letting them know that the submission was successful

3.1.2.5. Submitting Peer Reviews

Input: Student clicks “Download Instructions” button
Output: Review instructions are downloaded to the student’s computer
Input: Student clicks “Download Peer Review PDF” button
Output: The displayed PDF is downloaded to the student’s computer
Input: Student clicks the right arrow of the PDF display
Output: System displays the next page of the displayed PDF
Input: Student clicks the left arrow of the PDF display
Output: System displays the previous page of the displayed PDF
Input: Student clicks “Upload”
Output: The student’s file manager is displayed, allowing them to select a document from their hard drive

Input: Student selects a PDF to submit

Output: PDF is attached

Input: Student clicks on the “Score” field under “Score Submission”

Output: A list of numbers from 0 to 9 is displayed

Input: Student selects a number from the list

Output: The list disappears, and the selected number is displayed in the “Score” field

Input: Student clicks “Submit”

Output: Review is submitted; system sends an email to each member of the team letting them know that the submission was successful

3.1.2.6. Results Page

Input: Student clicks a course tab

Output: System displays a list of review results for the selected course

Input: Student clicks a review result

Output: Student is directed to the Peer Review Results page for the clicked review result

3.1.2.7. Peer Review Results

Input: Student clicks the “Download Reviews” button

Output: Reviews are downloaded to the student’s computer

Input: Student clicks on a review tab

Output: The review associated with the clicked tab is displayed

Input: Student clicks the right arrow of the PDF display

Output: System displays the next page of the review

Input: Student clicks the left arrow of the PDF display

Output: System displays the previous page of the review

3.1.3. Professor Interfaces

This subsection details possible interactions between the system and users in the professor role.

3.1.3.1. Main Navigation

Input: Professor clicks on the PeerSet Logo

Output: Professor is directed to the Home Dashboard
Input: Professor clicks “Courses & Assignments” tab
Output: Professor is directed to the Courses & Assignments page
Input: Professor clicks “Students & Teams” tab
Output: Professor is directed to the Student Information page
Input: Professor clicks “Quality Check” tab
Output: Professor is directed to the Quality Check
Input: Professor clicks their username
Output: A drop-down “sign out” button appears
Input: Professor clicks “sign out”
Output: Professor is signed out of PeerSet

3.1.3.2. Home Dashboard (Professor)

Input: Professor clicks an existing course under Manage Students & Teams by Course
Output: Professor is directed to the Student Information page, with the tab for the clicked course selected
Input: Professor clicks the “Create a New Course” button in the bottom right
Output: Professor directed to Create Course page
Input: Professor clicks “See All” button next to Quality Check
Output: Professor is directed to the Quality Check page
Input: Professor clicks on an individual review under Quality Check
Output: Professor is directed to the Quality Check page for the clicked review
Input: Professor clicks a course under Assignments by Course
Output: Professor is directed to the Courses & Assignments Page, with the tab for the clicked course selected
Input: Professor clicks an item under Assignments by Course
Output: Professor is directed to the assignment page for the clicked item

3.1.3.3. Create Course

Input: Professor clicks on the “Course Title” field
Input: Professor clicks “Independent” radio button
Output: This will adjust the course settings, so that every student has to submit their own assignment for the duration of the course

Input: Professor clicks “Randomized Teams” radio button
Output: This will randomly assign students into teams for the course
Output: A cursor appears, allowing the Professor to enter a title for the course
Input: Professor clicks the “Upload” button next to “Upload CSV”
Output: The professor’s file manager is displayed, allowing them to select a document from their hard drive
Input: Professor selects the appropriate CSV file
Output: CSV is attached
Input: Professor clicks “Cancel”
Output: Course creation is canceled; professor is directed to Courses & Assignments
Input: Professor clicks “Publish”
Output: Course is created; all students listed in the CSV file are automatically registered, and email notifications are sent to let them know they’ve been registered for a course

3.1.3.4. Courses and Assignments

Input: Professor clicks the “Create Course” button
Output: Professor is directed to the Create Course page
Input: Professor clicks a course tab
Output: A list of upcoming and past solution and review assignments is displayed for the selected course
Input: Professor clicks on an assignment for the selected course
Output: Professor is directed to the Assignment page for the clicked assignment
Input: Professor clicks “Create New Assignment”
Output: Professor is directed to the Build New Assignment page
Input: Professor clicks the trash can next to assignment.
Output: Delete Assignment prompt appears.
Input: Professor clicks Delete when prompted.
Output: Assignment is Deleted.
Input: Professor clicks View Student Info.
Output: Directed to Students & Teams Page.
Input: Professor clicks “Drafts” radio button
Output: Dashboard displays only assignments that have not been published

Input: Professor clicks “Active” radio button
Output: Dashboard displays only published assignments
Input: Professor clicks “All” radio button
Output: Dashboard displays all assignments

3.1.3.5. Build New Assignment

Input: Professor clicks on the “Title” field under Title Assignment
Output: A cursor appears, allowing the professor to enter a title for the assignment
Input: Professor clicks “Solution Due Date” field under Solution Assignment Content
Output: Calendar widget appears, allowing the professor to select a due date
Input: Professor clicks “Review Due Date” field Peer Review Assignment Content
Output: Calendar widget appears, allowing the professor to select a due date
Input: Professor clicks “Upload PDF file” button under Solution Assignment Content
Output: The professor’s file manager is displayed, allowing them to select a document from their hard drive
Input: Professor clicks “Upload PDF file” button under Peer Review Assignment Content
Output: The professor’s file manager is displayed, allowing them to select a document from their hard drive
Input: Professor clicks the “Preview” button
Output: Professor is directed to the preview page for the assignment
Input: Professor clicks “Save Draft” button
Output: System saves the assignment as a draft
Input: Professor clicks the “Publish” button
Output: The assignment is published; students receive an email notification letting them know that a new assignment has been created

3.1.3.6. Assignment Pages

Input: Professor clicks on the “Edit Assignment” button
Output: Professor is directed back to the Build Assignment page to make edits
Input: Professor clicks on the “Peer Review” tab on top of the PDF display

Output: Professor is directed to Peer Review PDF display
Input: Professor clicks on the “Solution” tab on top of the PDF display
Output: Professor is directed to Solution PDF display
Input: Professor clicks the right arrow of the PDF display
Output: System displays the next page of the displayed PDF
Input: Professor clicks the left arrow of the PDF display
Output: System displays the previous page of the displayed PDF

3.1.3.7. Student Information

Input: Professor clicks a course tab
Output: System displays a list of students enrolled in the selected course
Input: Professor clicks “Delete Course”
Output: A pop-up appears confirming the action, with “Cancel” and “Delete” buttons. If the Professor clicks the “Cancel,” button, then the action is canceled; if the Professor clicks the “Delete,” button, then the course is deleted
Input: Professor clicks “Add New Student”
Output: A form for adding new students appears at the top of the class list, which the professor can use to enter student information. Once the information is entered, the student is added to the course when the professor clicks the “Add” button
Input: Professor clicks the trash can icon next to a student’s name
Output: A pop-up appears confirming that the professor wants to delete the student, with “Cancel” and “Delete” buttons. If the professor clicks the “Cancel,” button, then the action is canceled; if the professor clicks the “Delete,” button, then the student is deleted
Input: Professor clicks the toggle button next to, “Student List”
Output: System displays a list of teams for the selected course
Input: Professor clicks the arrow icon next to a team
Output: The selected team list is expanded

3.1.3.8. Quality Check

Input: Professor clicks a course tab
Output: System displays a list of available submissions for the selected course
Input: Professor clicks a solution

Output: Professor is directed to the Solution Quality Check for the selected solution
Input: Professor clicks a review result
Output: Professor is directed to the Peer Review Quality Check for the selected peer review
Input: Professor clicks the "All" radio button
Output: All results from completed solution and reviews are displayed
Input: Professor clicks the "Needs Review" radio button
Output: Only results needing Professor review are displayed
Input: Professor clicks the "Completed" radio button
Output: Only results from completed reviews are displayed

3.1.3.9. Solution Quality Check

Input: Professor clicks "Download Solutions"
Output: All of the solution assignments are downloaded to the professor's computer
Input: Professor clicks the right arrow of the PDF display
Output: System displays the next page of the displayed PDF
Input: Professor clicks the left arrow of the PDF display
Output: System displays the previous page of the displayed PDF
Input: Professor clicks on a team name from the team list in the side panel
Output: The system displays the PDF solution submitted by that team
Input: Professor clicks "View Errors"
Output: A drop-down with two new buttons will appear; these buttons will be "Reupload PDF" and "Reject PDF." The errors that were found will also be displayed
Input: Professor clicks "Reupload PDF"
Output: The professor's file manager will appear, allowing them to select the corrected PDF for reupload
Input: Professor clicks "Reject PDF"
Output: The student will receive an email with a list of errors that need to be corrected
Input: Professor clicks on the "View Errors" icon while the drop-down is still displayed

Output: The Errors Found dropbox will disappear
Input: Professor clicks “Send Reviews”
Output: The solutions are sent out to teams for review

3.1.3.10. Peer Review Quality Check

Input: Professor clicks “Download Reviews”
Output: All of the peer reviews are downloaded to the professor’s computer
Input: Professor clicks on a team name from the team list in the side panel
Output: The system displays the PDF peer reviews for that team
Input: The professor clicks one of the review tabs at the top of the PDF display
Output: The system displays the review associated with the clicked tab
Input: Professor clicks the right arrow of the PDF display
Output: System displays the next page of the displayed PDF
Input: Professor clicks the left arrow of the PDF display
Output: System displays the previous page of the displayed PDF
Input: Professor clicks on the “Distributions” tab
Output: The grade distribution for the peer review assignment is displayed
Input: Professor clicks on the “Peer Reviews” tab
Output: A PDF display appears, showing the reviews for each team
Input: Professor clicks “View Errors” icon
Output: A drop-down with a “Reject PDF” button will appear. The errors that were found will also be displayed
Input: Professor clicks “Reject PDF”
Output: The student will receive an email with a list of errors that need to be corrected
Input: Professor clicks on the “View Errors” icon
Output: The Errors Found dropbox will disappear
Input: Professor clicks “Send Results” button
Output: Results are made available to students, and the professor is directed back to the Results page. Peer review is marked as completed, and will now show up as a completed review under the course Results tab

3.2. Functions

Logging In

1. If the user logs in via Google API, the software shall call the **getSpecificUser()** function to authenticate the user
    1.1. If the user's credentials are valid, they shall be logged into the software
    1.2. If the user's credentials are invalid, they shall be directed back to the login page, with a message letting them know that their credentials were invalid

Logging Out

1. If the user clicks the sign out dropdown in the top right corner, then the **getLogout()** function will be called, and the user will be redirected to the login page.

Creating a Course

2. When the professor creates a course, the **postCourse()** function will be called.
    2.1. The **postCourse()** function will call the **parse()** function to extract user information from the professor's CSV file
    2.2. The **postCourse()** function will then call the **addtoDatabase()** function to add the user information to the database
    2.3. Once the information from the CSV file has been called, the **postCourse()** function will call the **postStudent()** function to create student users and add them to the course
    2.4. If the professor edits a course, the **updateCourse()** function will be called to change course objects in the database
    2.5. If the professor deletes a course, the **deleteSpecificCourse()** function will be called to remove the course from the database

Document Distribution

3. If the professor decides to randomly generate teams for an assignment, the **makeTeamsAutomatically()** function will be called, and students will be randomly assigned to teams based on parameters set by the professor.
4. Once the teams have been created, the **addTeamsToDatabase()** function will be called. This will add each TeamID and an array of associated StuIDs to the database.

5. **checkIfStudentIsInDatabase()** looks to see if the student is in the database. If the student exists in the database, true is returned. If the student is not in the database, false is returned.

Creating an Assignment

6. When the professor wants to create an assignment the **postAssignment()** function will be called. Which will take the assignment description or upload a pdf of assignment with a dropbox and create it.
    6.1. If the professor needs to they can use the **updateAssignment()** function to change details of the assignment.
    6.2. If the professor deletes an assignment, the **deleteSpecificAssignment()** function will be called and the assignment will be removed.
7. When students submit their solutions to the assignment the **postSubmission()** function will be called to create the solution for professors to view using the **getSpecificSubmission()** function.
    7.1. The quality check object shall look for names of students in the course, profanity, an empty document.
8. When the professor has the solutions the **postAssignment()** function will be used to assign reviews to all teams(not including their own team).
9. When the professor distributes review assignments, the system will call the **postAssignment()** function to distribute solutions and grading rubric to teams for review.
    9.1. If the professor would want to change anything about the review assignment the **updateReview()** function would be called.
10. Then once the students have reviewed the solutions of the teams solutions they were given, the **postReview()** function will be used when students submit their reviews.
11. Once the **postReview()** function creates the assignment the professor can use the **getSpecificReview()** function to view the reviews.
    11.1. This quality check object shall do a second look for names of students in the course, profanity, an empty document.
    11.2. If the professor decides that a review should be discarded, **deleteSpecificReview()** shall be called and the review will no longer be visible

Quality Checking

12. The **QC()** function will check the PDF files for profanity, using a list of terms that can be edited by the professor. It also checks for the names of students enrolled in the course.

Viewing Results

13. When the professor views the grade distribution for a peer review, the **getQualityCheckedReviewsByProfessor()** function shall be called. This will allow the system to display the grades received by each team.
14. When the student views their received grades, the **getResultsAssignmentsByStudent()** shall be called. This will allow the system to display each of the grades received by the student, along with an average grade.

3.3. Performance Requirements

This application must have the ability to store an unlimited number of courses and assignments. The web application shall run on 2 servers: one for the database and one for the rest of the application. The application must be able to support unlimited simultaneous users. The application must be able to handle .PDF files, .CSV files, Word documents, and user credentials. There will be the use of JSON requests and responses.

3.4. Logical Database Requirements

- Database Tables:
    - User
        - Email
            - Type: varchar(32)
            - This will be the Users email.
        - Role
            - Type: char(len)
            - There shall be three user roles: Administrator, Professor and Student
                - Character allows for flexibility to accommodate more than two user roles
        - Settings

                - Type: char(len)
                - This value is a string representation of the settings for a course. The string will be length “len,” with each character in the string corresponding to a course setting.
            - UserID
                - Type: Primary Key integer
                - This value will be generated by the system and will be the primary value for identifying user objects
                    - This value will be independent of the StuID value for student users
    - Student
        - StuID
            - Type: varchar(32)
            - This value will be the student’s university ID number, which will be parsed from the CSV file on upload by the professor
        - FirstName
            - Type: varchar(32)
            - This would be a Student’s first name.
        - LastName
            - Type: varchar(32)
            - This would be a Student’s last name.
        - Email
            - Type: varchar(64)
            - This would be a Student’s email.
        - UserID
            - Type: Primary Key integer
            - This identifies the User.
    - Course
        - Title
            - Type: varchar(32)
            - This would be a string that is the title of the course.
        - Code
            - Type: varchar(32)
            - [TBD - may need to clarify with Engine]

        - sectionNumber
            - Type: varchar(32)
            - This value represents the section number for each course
        - Ends
            - Type: date
            - A date that will close submissions [may need further clarification from Engine]
        - Settings
            - Type: char(len)
            - This will identify a setting for the course.
        - CourseID
            - Type: Primary Key integer
            - This would be the Course ID. This value is shared by students enrolled in the course.
        - UserID
            - Type: integer
            - This integer is associated with the user that created the course
    - Assignment
        - CourseID
            - Type: integer
            - This ties the assignment to the course.
        - PDFDoc
            - Type: mediumblob (Binary Stream)
            - This will store a pdf document as a binary stream.
        - Settings
            - Type: char(len)
            - This will correspond to some setting for the assignment.
        - Title
            - Type: varchar(64)
            - This will be the title of the assignment.
        - isTeamed
            - Type: boolean

        - This boolean value will be “true” if an assignment is set to be completed in teams, and “false” if an assignment is set to be completed by individual students
    - isDraft
        - Type: boolean
        - This boolean value will be “true” if an assignment is saved as a draft, and “false” if an assignment is published
    - reviewStage
        - Type: boolean
        - This boolean value will be “true” if an assignment is currently in the peer review stage, and “false” if an assignment is still in the solution submission stage
    - assignmentID
        - Type: Primary Key integer
        - This is the identifier of the assignment.
- Submission
    - TeamID
        - Type: integer
        - Associates the submission with the team that made it
    - Signoff
        - Type: varchar(255)
        - This string is associated with student approvals of submissions
            - This is a concatenated string that adds information from each team member as they approve the submission (not sure which strings will be taken to form the concatenation - maybe StuID?)
    - PDFDoc
        - Type: mediumblob (binary stream)
        - This will store a pdf document as a binary stream.
    - SubID
        - Type: Primary Key integer
        - This value identifies the submission
    - submissionTime

            - Type: Date
            - This value represents the time of the submission
        - seen
            - Type: boolean
            - This boolean value will be "true" if a submission has been viewed by the professor, and "false" if the submission has not been viewed
        - comments
            - Type: longtext
            - This value represents any comments that are added to a submission
- Review
    - TeamID
        - Type: integer
        - Associates the submission with the team that made it
    - Signoff
        - Type: varchar(255)
        - This string is associated with student approvals of submissions
            - This is a concatenated string that adds information from each team member as they approve the submission (not sure which strings will be taken to form the concatenation - maybe StuID?)
    - PDFDoc
        - Type: mediumblob (binary stream)
        - This will store a pdf document as a binary stream.
    - RevID
        - Type: Primary Key integer
        - This value identifies the review
    - comments
        - Type: longtext
        - This value represents any comments that are added to a review submission
    - submissionTime

            - Type: Date
            - This value represents the time that the review was submitted
        - seen
            - Type: boolean
            - This boolean value will be "true" if a submission has been viewed by the professor, and "false" if the submission has not been viewed
    - Course_Team_Student
        - userID
            - Type: integer
            - This value associates a student user with a team (represented by the teamID value) within a course (represented by the courseID value)
        - courseID
            - Type: integer
            - This value associates a course with a team (represented by the teamID value) and the students on the team (with each student represented by a userID value)
        - teamID
            - Type: integer
            - This value associates a team with the students on the team (each represented by a userID value) and the course in which they are enrolled (represented by the courseID value)

3.5. Design Constraints

1. There must be help documentation for new users.
2. Users must be authenticated using JSON Web Tokens.
3. Open Liberty must be used as the host server of the application.
4. Eclipse Micro-Profile must be used in the development of the application.
5. All data must be saved to the database.
6. A microservice architecture must be used.
7. The application must be a three-role system, with administrator, professor and student.
8. Students must remain anonymous to other students.

9. Students cannot review their own solutions.
10. Distribution of review assignments must be as even as possible.
11. There will be the use of Google Login systems.
12. REACTJS being used for User Interface development.
13. Students must receive email notifications:
    a. When they are registered for a course
    b. When a new assignment sheet is posted
    c. When a new review assignment is posted
    d. When feedback becomes available

3.5.1. Standards Compliance

PeerSet must conform as closely as possible to WCAG 2.1, Level AA compliance within the limited development timeline.

Per U.S. Federal Code of Regulation 34 CFR Part 99, "Family Educational Rights and Privacy," students must remain anonymous during the CPR process because they are receiving a grade.

3.6. Software System Attributes

This section of this SRS document will contain Software System Attributes of which are Reliability, Availability, Security, Maintainability and Portability.

3.6.1. Reliability

After thorough testing and revisions, the software must not crash, have minimal vulnerabilities and have fast response times when given a request by users.

3.6.2. Availability

The services of this application shall be available at all times, even if one of the microservices is down, except during certain instances. The application may become unavailable if the servers, database or services are down for maintenance or troubleshooting.

3.6.3. Security

- Security shall not be held until the end of the project's development, and shall be in steady development throughout the project's whole development process.

- At this time, anybody can provide any email to register and login to the system.
- There will be 3 possible roles: professor, student or admin.
    - Users in the role of professor will have more privileges than users in the role of student. Administrators will install and run the software, and will have the ability to add professors to the database.
- JSON Web Tokens shall be used for authentication.

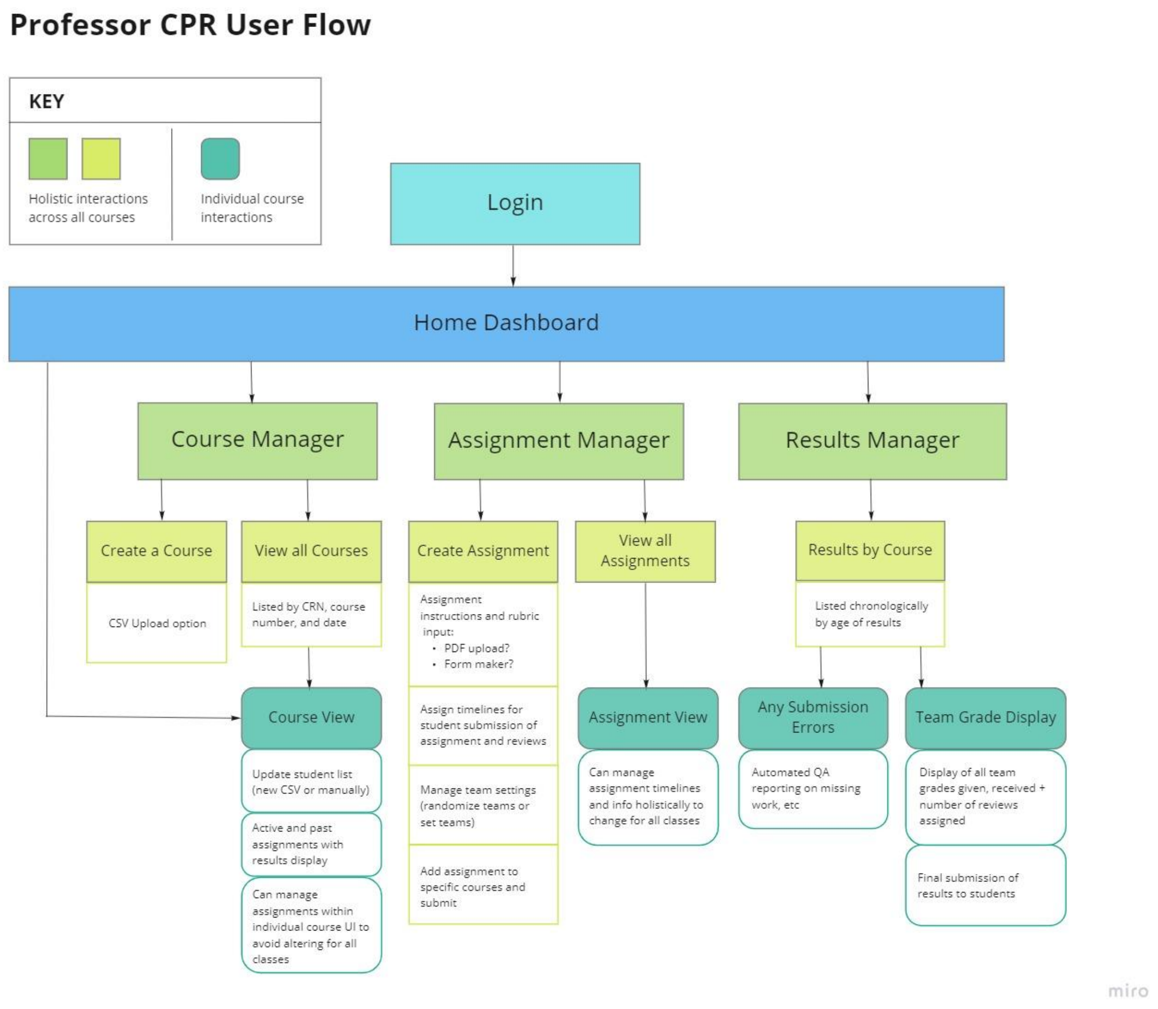
3.6.4. Maintainability

There will be periodic updates to the software during its 16 week development period. Security shall be developed throughout the project's development and not just at the end.

3.6.5. Portability

1. PeerSet, being a Web application, must be available for use on desktop or laptop computers running any operating system and any web browser.
    a. PeerSet must not be available to run on mobile devices such as tablets or smartphones
2. The code will be open source in the CSC480-21F main GitHub Repository, this source code can be cloned or downloaded.
3. PeerSet shall use portable languages, including ReactJS and Java

3.7. Diagrams



Professor CPR User Flow
KEY
Holistic interactions across all courses
Individual course interactions
Login
Home Dashboard
Course Manager
Assignment Manager
Results Manager
Create a Course
CSV Upload option
View all Courses
Listed by CRN, course number, and date
Create Assignment
Assignment instructions and rubric input:
• PDF upload?
• Form maker?
Assign timelines for student submission of assignment and reviews
Manage team settings (randomize teams or set teams)
Add assignment to specific courses and submit
View all Assignments
Results by Course
Listed chronologically by age of results
Course View
Update student list (new CSV or manually)
Active and past assignments with results display
Can manage assignments within individual course UI to avoid altering for all classes
Assignment View
Can manage assignment timelines and info holistically to change for all classes
Any Submission Errors
Automated QA reporting on missing work, etc
Team Grade Display
Display of all team grades given, received + number of reviews assigned
Final submission of results to students
miro

Figure 1: User flow diagram illustrating professor interactions with PeerSet

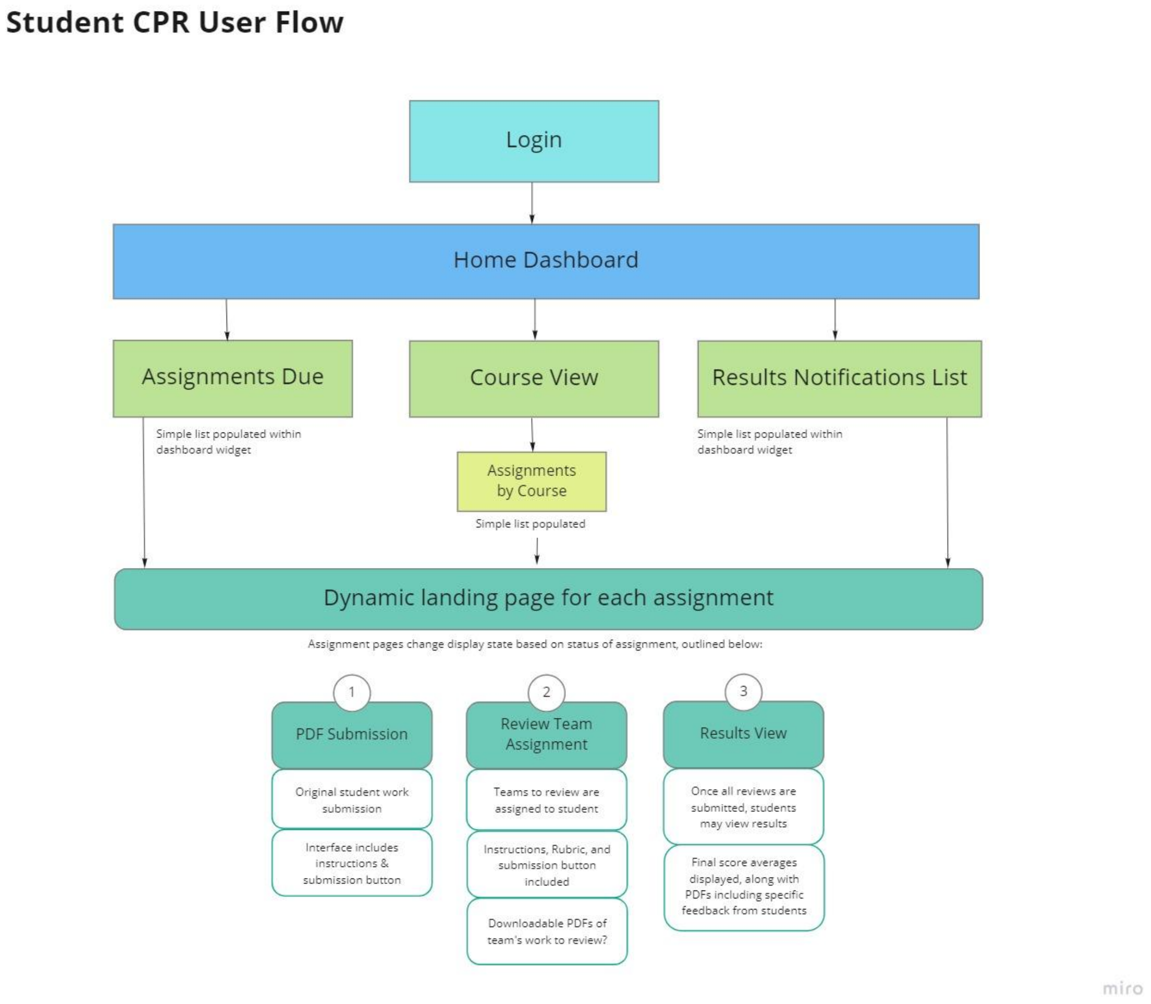

Student CPR User Flow
Login
Home Dashboard
Assignments Due
Simple list populated within dashboard widget
Course View
Assignments by Course
Simple list populated
Results Notifications List
Simple list populated within dashboard widget
Dynamic landing page for each assignment
Assignment pages change display state based on status of assignment, outlined below:
1
PDF Submission
Original student work submission
Interface includes instructions & submission button
2
Review Team Assignment
Teams to review are assigned to student
Instructions, Rubric, and submission button included
Downloadable PDFs of team's work to review?
3
Results View
Once all reviews are submitted, students may view results
Final score averages displayed, along with PDFs including specific feedback from students
miro

Figure 2: User flow diagram illustrating student interactions with PeerSet

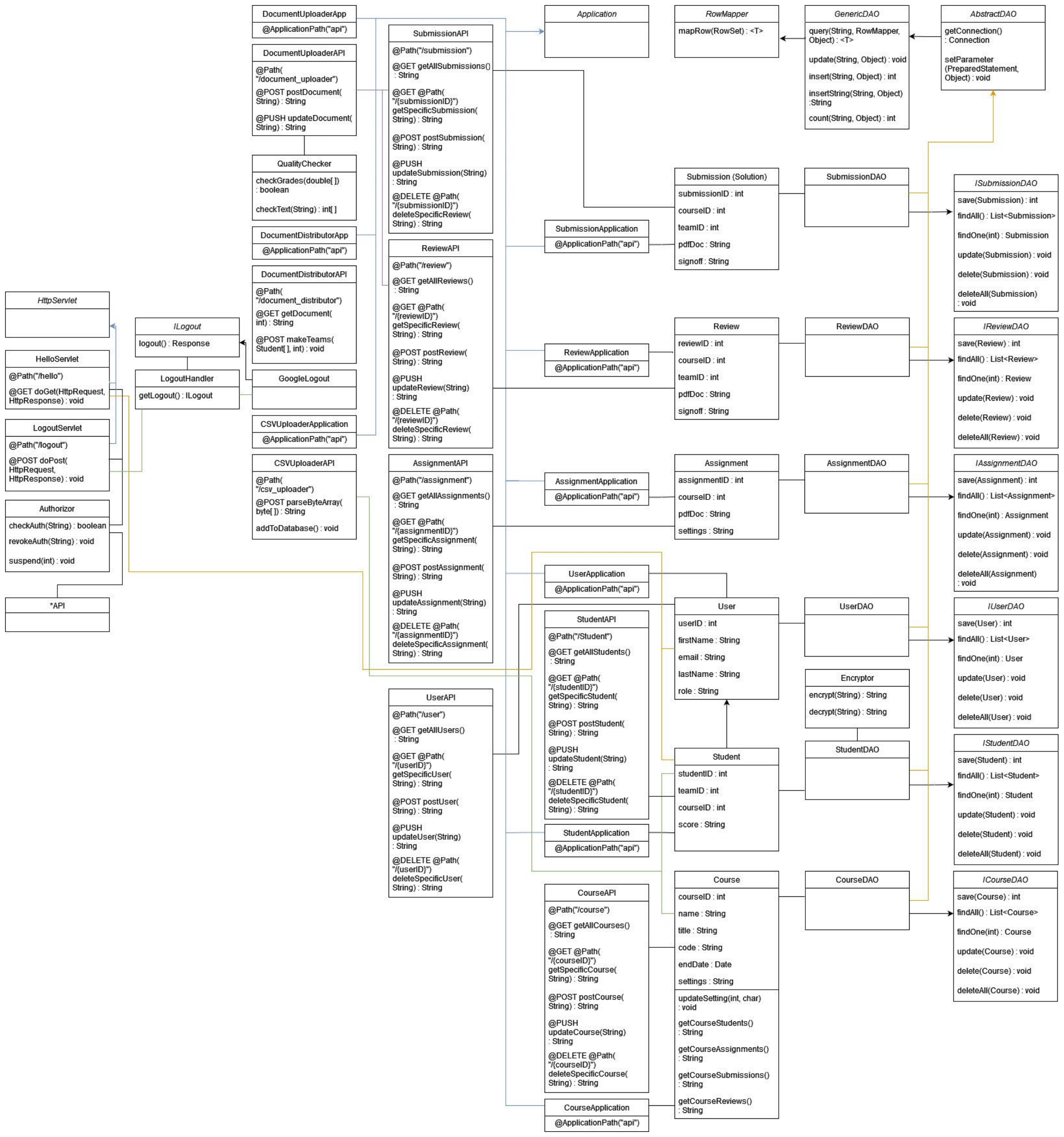

DocumentUploaderApp
@ApplicationPath("api")
DocumentUploaderAPI
@Path("/document_uploader")
@POST postDocument(String) : String
@PUSH updateDocument(String) : String
QualityChecker
checkGrades(double[ ]) : boolean
checkText(String) : int[ ]
DocumentDistributorApp
@ApplicationPath("api")
DocumentDistributorAPI
@Path("/document_distributor")
@GET getDocument(int) : String
@POST makeTeams(Student[ ], int) : void
GoogleLogout
CSVUploaderApplication
@ApplicationPath("api")
CSVUploaderAPI
@Path("/csv_uploader")
@POST parseByteArray(byte[ ]) : String
addToDatabase() : void
HttpServlet
ILogout
logout() : Response
LogoutHandler
getLogout() : ILogout
HelloServlet
@Path("/hello")
@GET doGet(HttpRequest, HttpResponse) : void
LogoutServlet
@Path("/logout")
@POST doPost(HttpRequest, HttpResponse) : void
Authorizor
checkAuth(String) : boolean
revokeAuth(String) : void
suspend(int) : void
*API
SubmissionAPI
@Path("/submission")
@GET getAllSubmissions() : String
@GET @Path("/{submissionID}") getSpecificSubmission(String) : String
@POST postSubmission(String) : String
@PUSH updateSubmission(String) : String
@DELETE @Path("/{submissionID}") deleteSpecificReview(String) : String
ReviewAPI
@Path("/review")
@GET getAllReviews() : String
@GET @Path("/{reviewID}") getSpecificReview(String) : String
@POST postReview(String) : String
@PUSH updateReview(String) : String
@DELETE @Path("/{reviewID}") deleteSpecificReview(String) : String
AssignmentAPI
@Path("/assignment")
@GET getAllAssignments() : String
@GET @Path("/{assignmentID}") getSpecificAssignment(String) : String
@POST postAssignment(String) : String
@PUSH updateAssignment(String) : String
@DELETE @Path("/{assignmentID}") deleteSpecificAssignment(String) : String
UserAPI
@Path("/user")
@GET getAllUsers() : String
@GET @Path("/{userID}") getSpecificUser(String) : String
@POST postUser(String) : String
@PUSH updateUser(String) : String
@DELETE @Path("/{userID}") deleteSpecificUser(String) : String
Application
SubmissionApplication
@ApplicationPath("api")
ReviewApplication
@ApplicationPath("api")
AssignmentApplication
@ApplicationPath("api")
UserApplication
@ApplicationPath("api")
StudentAPI
@Path("/Student")
@GET getAllStudents() : String
@GET @Path("/{studentID}") getSpecificStudent(String) : String
@POST postStudent(String) : String
@PUSH updateStudent(String) : String
@DELETE @Path("/{studentID}") deleteSpecificStudent(String) : String
StudentApplication
@ApplicationPath("api")
CourseAPI
@Path("/course")
@GET getAllCourses() : String
@GET @Path("/{courseID}") getSpecificCourse(String) : String
@POST postCourse(String) : String
@PUSH updateCourse(String) : String
@DELETE @Path("/{courseID}") deleteSpecificCourse(String) : String
CourseApplication
@ApplicationPath("api")
RowMapper
mapRow(RowSet) : <T>
GenericDAO
query(String, RowMapper, Object) : <T>
update(String, Object) : void
insert(String, Object) : int
insertString(String, Object) :String
count(String, Object) : int
AbstractDAO
getConnection() : Connection
setParameter (PreparedStatement, Object) : void
Submission (Solution)
submissionID : int
courseID : int
teamID : int
pdfDoc : String
signoff : String
SubmissionDAO
ISubmissionDAO
save(Submission) : int
findAll() : List<Submission>
findOne(int) : Submission
update(Submission) : void
delete(Submission) : void
deleteAll(Submission) : void
Review
reviewID : int
courseID : int
teamID : int
pdfDoc : String
signoff : String
ReviewDAO
IReviewDAO
save(Review) : int
findAll() : List<Review>
findOne(int) : Review
update(Review) : void
delete(Review) : void
deleteAll(Review) : void
Assignment
assignmentID : int
courseID : int
pdfDoc : String
settings : String
AssignmentDAO
IAssignmentDAO
save(Assignment) : int
findAll() : List<Assignment>
findOne(int) : Assignment
update(Assignment) : void
delete(Assignment) : void
deleteAll(Assignment) : void
User
userID : int
firstName : String
email : String
lastName : String
role : String
UserDAO
IUserDAO
save(User) : int
findAll() : List<User>
findOne(int) : User
update(User) : void
delete(User) : void
deleteAll(User) : void
Encryptor
encrypt(String) : String
decrypt(String) : String
Student
studentID : int
teamID : int
courseID : int
score : String
StudentDAO
IStudentDAO
save(Student) : int
findAll() : List<Student>
findOne(int) : Student
update(Student) : void
delete(Student) : void
deleteAll(Student) : void
Course
courseID : int
name : String
title : String
code : String
endDate : Date
settings : String
updateSetting(int, char) : void
getCourseStudents() : String
getCourseAssignments() : String
getCourseSubmissions() : String
getCourseReviews() : String
CourseDAO
ICourseDAO
save(Course) : int
findAll() : List<Course>
findOne(int) : Course
update(Course) : void
delete(Course) : void
deleteAll(Course) : void

Figure 3: UML diagram illustrating product functions

4. Supporting Information

4.1.1. MIT License