University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Developing a peer code review system for higher education

Mohammad U. Majid

June 25, 2021

Abstract

Peer code review is an established practice in software development and is gaining an increased interest in higher education. However, recent evidence suggests that the peer-review process suffers from two issues: low code review quality and a lack of student motivation. This dissertation aims to document the development of a peer code review system for higher education. The system aims to provide course instructors with the capability to create and conduct formative and summative peer code review exercises with students and meet the unique requirements of higher education. It also aims to improve student engagement with the review process and motivation to achieve better learning. Students rate the reviews they receive to provide feedback to the reviewer. The system awards badges to students for completing reviews and high-quality reviews. Based on a task-centric evaluation conducted with 17 students and 5 course instructors, it was found that the peer code review system scored in the top 10% on the System Usability Scale. In addition, students reported that they felt more engaged in the review process because of the rating system and more motivated because of the badge system.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Mohammad Usman Majid Date: 25 June 2021

Contents

1 Introduction

Peer review enables peers to critique and provide feedback on each others' work. In research, scholars scrutinise papers before they are published. In programming, developers review each other's code to identify defects, maintain a consistent coding style, and share knowledge. In education, course instructors create assignments, and students submit their work then review each other's work to provide feedback.

Peer review eases the administrative burden on staff in an educational setting and provides students with timely feedback on assignment submissions. Students provide constructive feedback to peers. They learn from receiving and writing peer reviews and can incorporate techniques learned from their peers in future assignments. Students also receive considerable exposure to different perspectives when reviewing work, which helps them reflect and improve their work. Overall, peer review in education is a positive experience for most students.

Peer code review (PCR) is an established practice in software development and is used to share knowledge, ensure high-quality code, and reduce code defects. However, it is not a standard practice in higher education but is receiving considerable attention as there are many learning benefits from student participation in the process (Lundstrom and Baker 2009).

1.1 Motivation

In order for higher education institutions to adopt PCR in classes, it is important that a potential PCR system meets the unique requirements of higher education. Course instructors use tools such as Aropa (2021) that are built for generic peer review. GitHub (2021) and similar tools are also used to facilitate code review style feedback.

Furthermore, there are two key issues with the PCR process: low code review quality and lack of student motivation. Student engagement with reviews and student motivation play an important role in determining the how successful the PCR process is.

With regard to review quality, Mulder et al. (2014) conducted a peer review study with 278 higher education students to examine student perceptions before and after experiencing peer review. The study showed that some students were concerned about the variation in review quality, especially when they put in great effort and received little in return. As a result, it revealed a level of dissatisfaction amongst students and decreased learning in the peer-review process. Wang et al. (2008) implemented a peer code review process with 9 level two students over a period of two academic years. They analysed student interviews to find that students rushed through code reviews and did not put in sufficient effort to review their peer's code. Wang et al. (2012) conducted a peer code review assessment with 87 level one students. They observed students submitting their reviews at the last minute with superficial or meaningless comments. Low code review quality can result in dissatisfaction with the peer review process, as it has an impact on the student's learning experience for both the reviewer and code author. There is, therefore, a need to improve student engagement with the review process to ensure higher quality reviews.

Turning now to student motivation, Turner et al. (2011) carried out a peer code review study with 134 students from two universities. Over one semester, students completed an attitudinal survey before and after four assessed PCR assignments. They noted that students showed a lack of motivation, and this resulted in them missing out on a valuable learning opportunity. In addition, Indriasari et al. (2020) identified a lack of motivation as a barrier to successful PCR in education. Low motivation can lead to students avoiding or delaying their PCR tasks. It can also affect student engagement with the review process. So, there is a need to improve student motivation to ensure students learn as much as possible from the PCR process.

1.2 Aims

The aim of this dissertation is to document the development of a PCR system for higher education. The PCR system must enable course instructors to create and conduct peer code review exercises with students. In addition, the PCR system must be compatible with the needs of higher education. As indicated previously, the PCR system also aims to improve student engagement with the peer review process and increase student motivation to achieve better learning.

1.3 Chapter Summary

Thus far, this dissertation has introduced peer code review in education and given the reader some intuition about why such a system would be helpful for higher learning institutions. It then outlined two challenges in the PCR process in the context of higher education. Finally, it listed the aims and objectives of this project.

The rest of this dissertation is structured as follows: The next chapter summarises relevant research presenting background information. It introduces and outlines key features in related products and proposes a peer code review system. Chapter 3 lists the functional and non-functional system requirements. Chapter 4 describes the design of the system using a top-down approach. Chapter 5 outlines the development process and implementation of key features. Chapter 6 outlines the evaluation of the system through testing and a user evaluation. Finally, Chapter 7 summarises this dissertation and proposes suggestions for future work.

2 Background

This chapter presents relevant background research surrounding peer code review. It then introduces and outlines key features in related products. Lastly, it proposes a peer code review system that meets the aims and objectives outlined in Section 1.2.

2.1 Background Information

Peer review systems used in higher education involve two types of users: course instructors and students. Instructors set up peer-review assignments. Students complete the assignments by submitting their work and reviewing their peer's work. Some systems support summative assignments, wherein students revise and submit their work again after reflecting on received feedback. During the review process, students act as authors, reviewers, and revisers, depending on the stage and type of assignment.

Peer feedback provides benefits to both the reviewer and the author. The quality of feedback a student receives can impact their learning experience. Cho et al. (2006) and Gielen et al. (2010) show that peers are capable of providing feedback of equal value to that provided by course instructors. Reily et al. (2009) found that aggregating multiple student reviews improves the overall review accuracy. Receiving more reviews is more likely to ensure students get the feedback they need. Pearce et al. (2009) suggest at least 2-3 reviewers per assignment. Song et al. (2020) found that obtaining five reviews for each student submission increased review accuracy and created grades similar to that of teacher assistant reviews. These studies highlight the importance of multiple student reviews in higher education.

The peer-review process can reveal the identity of the code author, the reviewer, or both. Hiding the identity of the reviewer allows peers to deliver more critical feedback (Lu and Bol 2007). Anonymity ensures all personal relationship factors are excluded, but does not allow the author to ask the reviewer for clarification. Morales-Martinez et al. (2020) found that in student review tasks, anonymity improved participation and reduced biases. These studies demonstrate that anonymity improved the peer-review process as a whole.

During the peer review process, an evaluation rubric is used by students to review their peer's work. The evaluation rubric is defined by the course instructor for an assignment and consists of a series of questions that provides structure to peer reviews and represents the assignment's assessment criteria. Peer review guided by a rubric has more potential for learning as rubrics make assessment expectations and criteria explicit (Song et al. 2020). Students can also use a rubric to mark their peer's work. Panadero et al. (2013) found peer assessment with a rubric was more valid and reliable.

With regards to review quality, a good peer code review must be fair and accurate. Hundhausen et al. (2013) conducted four code review studies with level one university students. They used their findings to establish a set of ground rules for PCR. One rule states that reviews should identify strengths to provide the student with positive reinforcement. Another rule mentions that reviews should focus on improvements, ideally phrased in positive action-oriented terms to be well-received. A good quality code review in higher education must identify strengths and weaknesses, provide actionable improvements, and be fair and accurate.

Regarding student motivation, game-based learning has been shown to enhance learning and motivation through the introduction of game elements (McGonigal 2011). Reward-based gamification can be particularly effective in teaching skills with real-world value (Nicholson 2015). Once students understand the value, rewards are no longer needed. However, rewards must be small since extrinsic rewards can undermine intrinsic motivation. In higher education, the introduction of a points or badge based incentive to peer code review has potential to improve and maintain motivation.

2.2 Related Products

Before proposing a peer code review system, it is necessary to explore related products and highlight their key features. This section discusses GitHub, Peergrade, and Aropa. It identifies potential opportunities for the PCR system. It is important to note that none of the products discussed below attempt to improve student engagement with the review process or increase student motivation.

2.2.1 GitHub

GitHub (2021) is a web-based source code management system used by millions of software developers worldwide and over 8,000 companies. It provides advanced features for reviewing and commenting on code submitted by others. However, it does not provide functionality for peer code review in education. There is no anonymity, no automatic reviewer designation, and no option to postpone reviewer feedback to deliver all at once after a review deadline.

Features:

- Students can upload code to a public or private repository.
- Anyone with access to the repository can review the code.
- Students can make changes to their code and request further feedback from reviewers.
- Many people can review code and discuss improvements openly.
- Reviewers can leave comments on any line in the code.
- Basic features are free for all individuals and organisations.

GitHub provides many additional features and has extensive support for code reviews. However, these features are limiting for peer code review in an educational setting.

2.2.2 Peergrade

Peergrade (2021) is an all-in-one web-based peer review platform used by over 8,000 institutions with over 100,000 monthly active users. It enables instructors to use peer review with students using an instructor defined rubric.

Firstly, instructors create an assignment and an evaluation rubric. Students upload their work to Peergrade, and after the hand-in deadline, work is distributed amongst peers for review. Students complete peer reviews, and after the deadline, feedback is returned to students.

Features:

- Platform supports peer review of all types of work.
- Students can submit work individually and in groups.
- Students can grade each other's assignments through the rubric.
- Instructors can view student initial submissions and reviews.
- Instructors can enable or disable anonymity for assignments.
- Public rubric library and institutional rubric sharing for fast rubric creation.

- Data-driven insights into courses, assignments and rubric quality.

Peergrade provides a vast array of features for students and instructors. However, it does not provide support for revised final submissions and summative assignments. It is also costly for institutions. Peergrade plans charge per student per year. Lastly, it is limiting for institutions as they cannot customise or modify the system.

2.2.3 Aropa

Aropa (2021) is a web-based peer review platform developed by Dr John Hamer and Dr Helen Purchase, academics at the University of Glasgow. The peer-review process is similar to Peergrade. Peers submit their work and complete reviews. The platform supports instructor defined rubrics, anonymity, and summative assignments. Aropa supports fifteen different assignment configuration options. Purchase and Hamer (2018) analysed the use of the different assignment configurations in over one thousand Aropa peer-review assignments. They found that peer review assignments are set up in very different ways.

Features:

- Students can provide qualitative and quantitative feedback to their peers.
- Reviewer allocations can be made manually or automatically with options for adjustments.
- Available option to utilise tutors in parts of the review process.
- Available option to award marks for submission quality, where quality is determined through peer assessment via the rubric.
- Available option to award marks for completing reviews and review quality, where quality is marked manually via tutors or instructors.
- Free for institutions, since 2009.

The list of features above highlights a few of Aropa's features and is by no means exhaustive. However, Aropa is built for general peer review and, therefore, to utilise Aropa for peer code review, students must place their code into a file and upload the file to Aropa. In addition, Aropa does not attempt to improve student engagement or motivation, factors which are both crucial to the students' learning experience.

2.3 Proposed System

Based on research, the review of related products, and this project's aims, a peer code review system for higher education is proposed.

The proposed system enables peer code review in higher education and is focused on improving student engagement with the review process and student motivation. It enables the creation of courses, assignments and instructor-defined rubrics. The system utilises double-blind peer review to provide students with anonymity in all stages of the process. An assignment consists of several stages in a fixed order. Each stage has a deadline to ensure all students receive feedback simultaneously and have equal time to complete their work.

Students first submit their initial code, then review three of their peer's code. Once the review deadline has passed, feedback is released to students. Three reviews ensure students get the feedback they need, and the overhead for students is not a burden. An additional stage is proposed wherein students rate the reviews they receive, intending to improve student engagement with the review process. Students rate reviews based on four categories: accuracy, fairness, identification of strengths, and actionable improvements. Once the rate review deadline has passed, review ratings are released to reviewers. Assignments can be formative or summative. Summative assignments involve a further stage wherein students submit their revised code after the peer-review process.

Students must submit their code to review their peer's code. Without submitting the code, students cannot receive or rate reviews. Reviewers are assigned after the initial code submission deadline.

Assignment weights can be configured to specify how the student's work is graded. Students can receive marks for submitting code and reviewing their peer's code. Rubric questions can contain marks that allow for code quality to be numerically computed. Similarly, rating reviews allows for review quality to be numerically computed. Students can receive marks for rating their peer's reviews as an incentive. In the case of summative assignments, a student can receive marks for submitting revised code. The weight of the final submission on the assignment grade and the number of marks it is marked out of is also configurable. Instructors mark and provide feedback on final student submissions.

In addition to the rate review stage, a game-based incentive is proposed wherein students are awarded badges for completing reviews and high-quality reviews, intending to improve student motivation. Students are awarded bronze, silver, and gold for high-quality reviews.

2.4 Chapter Summary

This chapter has discussed relevant background research and key features in GitHub, Peergrade, and Aropa to help understand requirements and propose a peer code review system for higher education.

In the chapter that follows, the requirements of the PCR system are summarised and listed using a prioritisation technique.

3 Requirements

Requirements gathering is key in every Software Engineering project. However, requirements must be managed and prioritised carefully, as all projects have finite resources. This chapter lists the functional and non-functional requirements of the PCR system that were identified in the previous chapter.

3.1 MoSCoW Prioritisation Technique

As far as prioritising requirements is concerned, there are several accepted techniques. The MoSCoW technique classifies each requirement into one of four prioritisation categories (Storer 2019).

1. Must have requirements are non-negotiable MVP requirements. Without them, the system is unusable.
2. Should have requirements are important and painful to leave out. However, without them, the system is still viable.
3. Could have requirements are desirable, but not important.
4. Would like to have requirements are out of a project's scope.

3.2 Functional Requirements

There are two types of requirements: functional and non-functional. Functional requirements capture the features required in a software system. Non-functional requirements represent the qualities a software system must possess. Before listing the non-functional requirements, this section prioritises the functional requirements of the PCR system using the MoSCoW prioritisation technique.

In terms of users, the PCR system must have two types of users: instructors and students. However, it must also have administrators as they are needed to add instructors to the system. The system must allow all users to sign in, sign out, change and reset their password via email.

3.2.1 Instructor

What follows are 28 instructor requirements categorised using the MoSCoW prioritisation technique.

Must Have:

1. Create and view courses.
2. Add students, that are known or unknown to the system, to a course.
3. Remove students from a course.
4. Create and view formative and summative assignments.
5. Create and view rubrics.

6. Add and remove questions within a rubric.
7. Mark and give feedback to a student's final submission.
8. Modify marks awarded and feedback given to a student's final submission.
9. Download student final marks with results breakdown.
10. Set assignment weights, including initial code submission participation and quality, reviews participation and quality, and rate reviews participation. For example, a formative assignment could award a student 20% of the available marks for submitting the initial code, 10% for the code quality (computed using reviews received on a student's code), 40% for submitting three reviews, 20% for the review quality (computed using ratings received on a student's reviews), and 10% for rating received reviews.
11. Set additional weights for summative assignments, including final code submission participation and quality, and the number of marks final submission is marked out of. For example, a summative assignment could award 0% for submitting the final revised code, 100% for the quality, which the instructor marks out of 40 marks.
12. Set assignment release date and deadlines, including initial code submission deadline, complete reviews deadline, rate received reviews deadline, and final code submission deadline if the assignment is summative.

Should Have:

13. Modify and delete courses.
14. Modify and delete assignments before the start date.
15. Modify and delete an ongoing assignment.
16. Modify and delete rubrics that are not in use.
17. Modify and delete rubrics in use for ongoing assignments.
18. Re-order questions within a rubric.
19. Set weightings for questions in a rubric to allow code quality to be computed.
20. Add and remove students from a course in bulk.

Could Have:

21. Publish results within the system.
22. View which students have joined the system for a given course.
23. Ability to format assignment task description.

Would Like To Have:

24. View student initial submissions.
25. View student reviews.
26. View student review ratings.
27. Choose the number of reviewers required for an assignment.
28. Add multiple-choice questions to a rubric.

3.2.2 Student

Moving on now to consider students, this section categorises 18 student requirements using the MoSCoW prioritisation technique.

Must Have:

1. Join the system via an email invite.
2. View courses.
3. View assignments.
4. View assignment details including structure, task description, deadlines, and the weights assigned to each stage of the assignment.
5. Submit code before the initial submission deadline.
6. Review assigned submissions using the instructor defined rubric before the review deadline.
7. View reviews received from peers after the review deadline.
8. Rate reviews received, giving feedback to peers before the rate reviews deadline.
9. Submit revised code before the final submission deadline.
10. View all badges awarded from completing reviews on badges page.

Should Have:

11. Give marks to a peer's submission to allow their code quality to be computed if the assignment has an initial code submission quality weight and rubric questions have weightings.
12. Modify code submissions, reviews and ratings before the deadlines.

Could Have:

13. View results on the system.
14. View badges awarded from completing reviews in the top bar.
15. Receive an email when added to a course.

Would Like To Have:

16. When reviewing code, allow for commenting on specific lines.
17. View a summary of upcoming deadlines.
18. Code editor for uploading code.

3.2.3 System

Before proceeding to list the non-functional requirements, this section categorises 7 additional system requirements using the MoSCoW prioritisation technique.

Must Have:

1. Invite students to join the system via email when they are added to a course for the first time.
2. Hide student details including name and ID from peers.
3. Automatically allocate reviewers randomly, immediately after the assignment submission deadline.
4. Time zone support.

Could Have:

5. Dark mode toggle.

Would Like To Have:

6. Algorithms for reviewer designation.
7. Automatic marking for correctness, e.g. run code against a test suite.

3.3 Non-Functional Requirements

Moving on now to consider the non-functional requirements. These requirements represent the qualities a software system must possess and can be critical to the product's success. For example, if a system is too slow, it can fail to satisfy user needs.

Performance - the system must be responsive and must return results in a timely manner.

Security - the system must be secured with authentication and use appropriate access control mechanisms to ensure data confidentiality and integrity. For example, the system must prevent a student from modifying their reviews after the deadline.

Availability and scalability - the system must be accessible to users and not experience any downtime due to traffic. Therefore, it must be possible for the system to scale when deployed with a load balancer or on modern cloud-based solutions, allowing the system to handle increased load effectively.

Usability - users must be able to use and navigate through the system with ease effectively.

Reliability - the system must be reliable. If an action fails, it must handle the error and display an appropriate error message.

Compatibility - the system must be accessible on devices of varying sizes. It must be friendly for desktop, tablet and mobile use.

3.4 Chapter Summary

This chapter prioritised the functional requirements of the system for instructors, students and the system itself using the MoSCoW technique. Subsequently, it listed the non-functional requirements of the system.

In the chapter that follows, the design of the PCR system is presented.

4 Design

The design of the PCR system was crucial to adhere to the many requirements identified in the previous chapter. In the following pages, a top-down approach to design the PCR system is presented. Firstly, it identifies the different components required through a system architecture diagram. Subsequently, it discusses the behaviour of the system and its users in a peer code review exercise. Then, it presents the initial prototype designed using wireframe diagrams. Lastly, it concludes by presenting the design of the Backend API endpoints and the database.

4.1 System Architecture

The system architecture is the most crucial part of the PCR system. A well-thought-through design ensures that non-functional requirements, such as performance, scalability, and security, are built into the system. The architecture can determine the quality and longevity of the system. A poor architecture design decision early on can cause significant issues later on during implementation. Carefully designing the system architecture is an investment that makes code easier to maintain as its structure is visible and creates a clear separation of concerns between sub-systems (Stromeyer 2017).

The architecture of the PCR system was designed carefully to ensure high-quality software that can meet all its functional and non-functional requirements. The system must be usable on mobile, tablet, and desktop. The architecture must allow cross-platform compatibility with minimal effort as project resources are limited. Therefore, a web-based architecture is employed.

The architecture of the PCR system, shown in Figure 4.1, resembles a client-server web application. The architecture comprises a client-side and a server-side. The client-side consists of the user's web browser and a Frontend User Interface. The server-side comprises a Backend Server with three parts: an API exposing REST endpoints, the business logic of the system, and a data model for interacting with the database. It also shows four additional components connected to the Backend Server. These components comprise an email server, a message broker and worker nodes, and a relational database. The diagram also shows the interactions between different components. For example, the client and server sides interact and communicate through a request-response cycle.

It is important to note that only the Backend Server and the Frontend User Interface are project deliverables. Nevertheless, the remaining components are discussed in this dissertation as they are needed and utilised by the Backend to meet the functional and non-functional requirements. The following two sub-sections discuss the components in the system architecture in further detail, explore the way in which they relate to the system requirements, and justify the need for the database, email server, message broker and worker nodes.

4.1.1 Deliverable components

The Frontend renders the user interface to the browser via client-side rendering. Client-side rendering prevents unnecessary full-page reloads, provides a smooth user experience with rich interactions and enforces separation of concerns between the business logic on the Backend

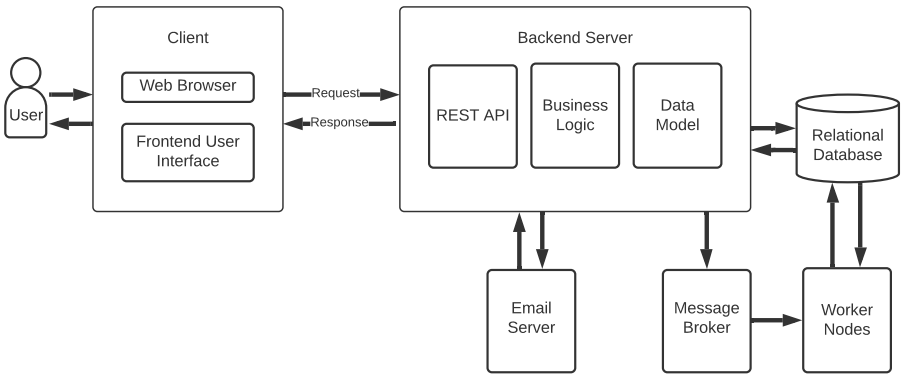


Figure 4.1: *A system architecture diagram, showing the interaction between different components on the client and server-side of the PCR system. It shows the Frontend User Interface and the Backend Server, the two project deliverables. It also shows the Backend Server utilising several third-party non-deliverable components to meet system requirements.*

and the user interface. Furthermore, client-side rendering allows for static files to be hosted via content distribution networks, allowing for scalability, a key non-functional requirement of the system. Additionally, client-side rendering lightens the load on the Backend, which enables it to serve responses faster, ensuring a more responsive system.

The Backend consists of three sub-components. The data model provides an interface to a relational database, which is used to store persistent data. The business logic implements the system requirements. The REST API exposes the system's business logic via HTTP GET, POST, PATCH, and DELETE endpoints.

The Frontend communicates with the Backend Server via the REST API. This design allows both sides to be developed with great flexibility in the choice of development frameworks. It becomes easier to work on one side of the system while the other side remains untouched. In addition, it becomes simpler to switch frameworks in the future. There are several additional benefits to designing the Frontend and Backend in this fashion (Joshi 2018).

4.1.2 Non-deliverable components

A database is needed to allow the Backend Server to store, access, update and delete persistent data. A relational database is used as data points are related. For example, one assignment can have many student code submissions.

An email server is used to send and retrieve emails. Two key requirements of the PCR system indicate the need for an email server. Firstly, the system needs to invite students to join when they are added to a course for the first time. Secondly, the system needs to provide password reset functionality via email. However, sending emails is a CPU-intensive job for the Backend. It can take longer than expected for many reasons, including network faults.

Web applications work in a request-response cycle. Long-lasting tasks such as sending emails can reduce system performance significantly. When adding a student to a course, an instructor may have to wait a long time while the Backend communicates with and sends an email via the email server. However, as identified in Chapter 3, performance is a key non-functional requirement; the system must be responsive and return results in a timely manner.

In addition, the PCR system needs to handle students being added to the system in bulk. This means that the system must be able to cope well when it must send hundreds of emails. It cannot

send all the emails at once as each invitation email is unique. Furthermore, automatic reviewer designation after the code submission deadline is another example of a long-lasting task.

Task delegation, is therefore, required to allow the Backend to delegate long-lasting tasks. It enables the PCR system to remain responsive and available as time-consuming tasks are executed in the background. Task delegation is achieved through a message broker and worker nodes. The Backend generates and places tasks onto a task queue via a message broker. Following the producer-consumer pattern, worker nodes are child processes that consume and execute individual tasks from the task queue.

4.2 Activity Diagram

Before presenting an initial prototype, it is necessary to understand the behaviour of a peer code review exercise with the rate review stage. The activity diagram, as illustrated in Appendix B, shows the behaviour of the PCR system and its users before and during a peer code review exercise. Visually, we can differentiate the behaviour of administrators, instructors, students as authors and reviewers, and the system itself. The diagram shows the following behaviour:

1. An administrator creates an instructor account.
2. The instructor creates a course and adds students to the course.
3. The system invites students that do not have an account to join the system.
4. Students join the system.
5. The instructor creates a rubric and a summative assignment.
6. Students as authors submit their code before the submission deadline.
7. The system randomly assigns students to review code.
8. Students as reviewers review their peer's code before the review deadline.
9. Students as authors view the reviews they received after the review deadline.
10. Students as authors rate the reviews they received before the rate review deadline
11. Students as reviewers view the ratings they received after the rate review deadline.
12. Students as authors submit their final revised code.

4.3 Initial Prototype

During the design process of the PCR system, an initial prototype was developed, refined, and then used to influence implementation. The prototype comprises 22 wireframe diagrams that represent instructor, student and authentication functionality.

A wireframe diagram is a simple and clear sketch that represents part of a system. Wireframe diagrams depict the different elements and functionalities on a page. They are easy to draw, understand, modify, and require no coding. They allow the design of a system to be refined and problems to be caught prior to implementation.

Figure 4.2 is a wireframe diagram depicting the design of the instructor courses page. This page allows instructors to create new courses and manage existing courses. For each course, an instructor can create, edit and delete existing assignments, manage students and the course itself. Under each course, there are details of each assignment, including the assignment title, weight and type. It also has options to download and publish results for completed assignments.

Figure 4.3 is a wireframe diagram, depicting the peer code review page. This page allows a student to review their peer's code. It displays the code which the student must review and enables the student to complete the review by following the rubric. In addition, the rubric allows the student to assess their peer's code.

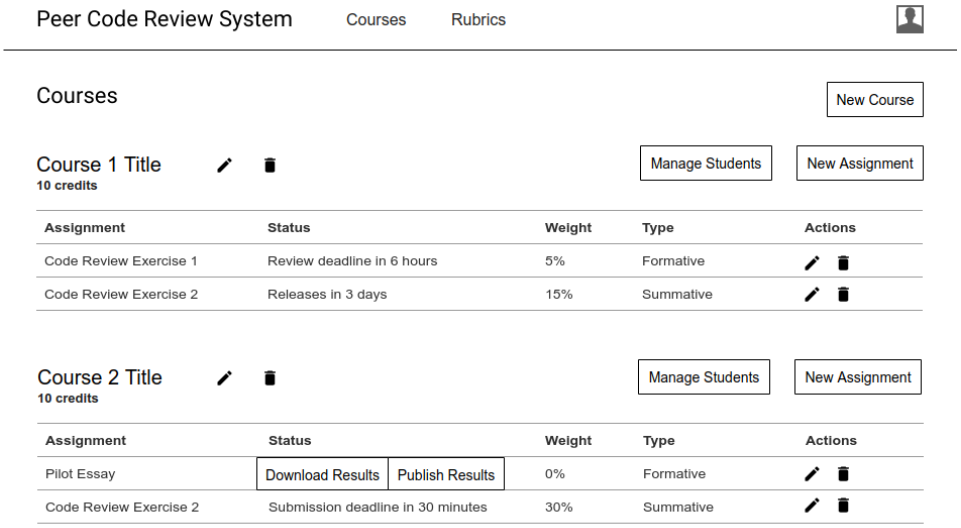


Figure 4.2: *The design of the instructor courses page. On the page, an instructor can create new courses and manage existing courses individually. For each course, an instructor can create, edit, and delete existing assignments, manage students and the course itself. For each course, the assignment details are shown below. It also shows one completed assignment, which provides options to download or publish results.*

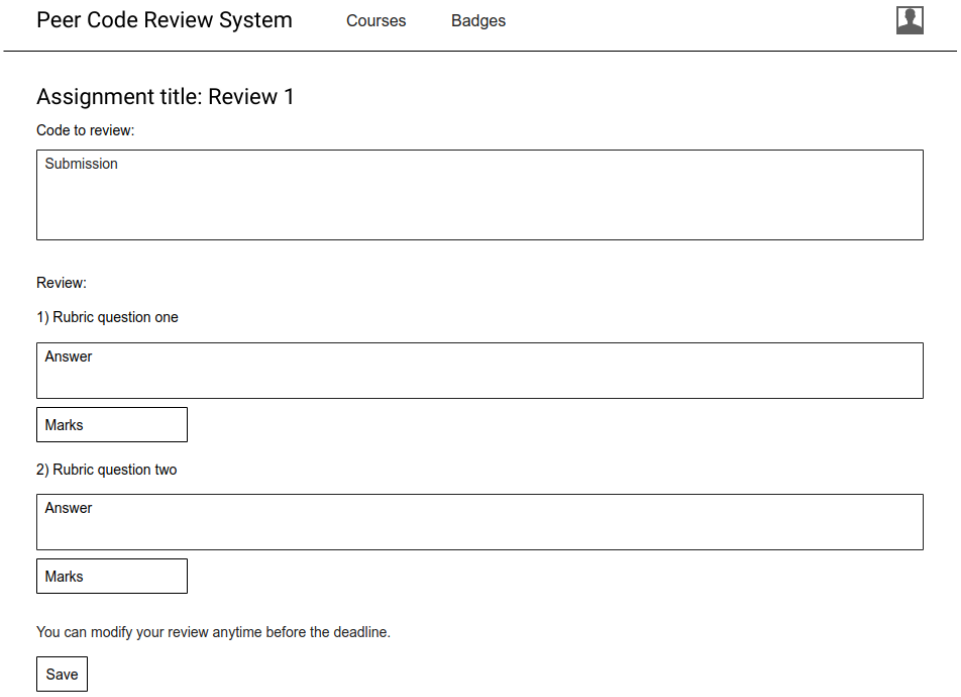


Figure 4.3: *The design of the student review page. On the page, a student can see the code they must review and can complete the review by following the assignment rubric.*

In addition, 4 wireframes, shown in Appendix C, were created to represent authentication functionality for all users. Then, 12 wireframes, shown in Appendix D, were created to represent all instructor features of the system, including creating and adding students to courses. Lastly, 6 wireframes, shown in Appendix E, were created to represent all student features, including partaking in a peer code review assignment and badges.

4.4 Backend API Endpoints

The Backend REST API exposes the system's business logic via HTTP GET, POST, PATCH, and DELETE endpoints. During the design of the PCR system, 47 Backend API endpoints were designed which would provide all the functionality that the Frontend would need from the Backend. The endpoints were designed with access control mechanisms to ensure security, a key non-functional requirement. The following two endpoints illustrate this:

Create Course Endpoint: Sending an HTTP POST request to */api/courses/* with a course in the request body creates a course. However, only instructors can access this endpoint to create a course.

Delete Assignment Endpoint: Sending an HTTP DELETE request to */api/assignments/ "assignment_id"* with the assignment ID in the request URI deletes an assignment. In the same way as the endpoint above, only instructors can access this endpoint. However, only the course instructor, who created the assignment, can access this endpoint to delete the assignment. In general, the API endpoints were designed to ensure that data can only be modified by its respective owner.

Furthermore, to ensure data integrity, code submissions, peer reviews and review ratings can only be saved if the assignment deadlines allow it. For example, students can only make changes to their submissions if the assignment submission stage is open.

All unsuccessful requests contain an appropriate error message and code. This provides the Frontend with the necessary information to handle and display an appropriate error message, ensuring reliability, a key non-functional requirement. For example, attempting to retrieve an assignment that does not exist returns an HTTP 204 No Content with the error message "Assignment does not exist". In addition, if a user is denied access to an endpoint, HTTP 403 Forbidden is returned to indicate that the endpoint exists but refuses to authorise the user.

In addition, 6 authentication endpoints, shown in Appendix F, are accessible to both students and instructors to sign in, sign out, change and reset passwords. An additional 17 student endpoints, shown in Appendix H, are accessible only to students to get assignments, submit code, complete reviews, etc. Lastly, 24 instructor endpoints, shown in Appendix G, are accessible only to instructors to manage courses, assignments, students, and more.

4.5 Data Model and Database Design

The data model provides an interface to the database and further encapsulates the interaction between data. For example, the data model ensures that initial code submissions cannot be saved when the assignment has not been released. The underlying database only enforces database constraints, such as unique primary key constraints.

The design of the database is imperative to the performance of a system. Using the Crow's Foot notation, the entity-relationship (ER) diagram presented in Figure 4.4 shows 11 database tables. Each table has a unique primary/composite key and several non-prime attributes.

For example, to store rubrics and their questions in the database, the ER diagram proposes two tables, Rubric and RubricElement. The rubric table has four fields: rubric ID (primary key, integer), instructor ID (foreign key, varchar), title (varchar), and description (varchar). The

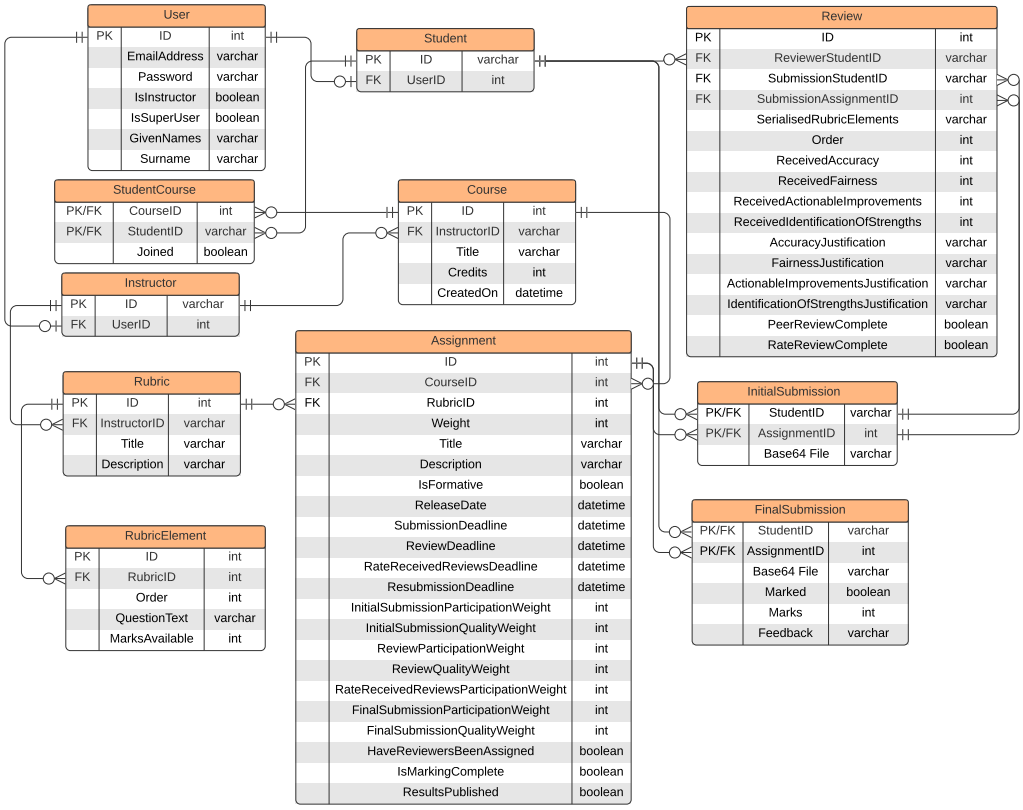


Figure 4.4: *An entity relationship diagram showing the tables in the database of the PCR system. Each table contains several attributes and their data types. Relationships between tables are represented using foreign key relationships, in Crow's Foot notation.*

rubric element table has five fields: rubric element ID (primary key, integer), rubric ID (foreign key, integer), order (integer), question text (varchar), marks available (integer). This design allows a rubric to belong to one and only one instructor and contain zero or more questions. Each question in a rubric has an order and a number of marks available to allow question re-ordering and peer assessment.

The relational schema is highly optimised, so data required by the Backend can be retrieved as quickly as possible, minimising the number of tables accessed. The design conforms to functional dependency and normalisation theory to ensure data integrity, minimum redundancy and efficiency.

4.6 Chapter Summary

This chapter took a top-down approach to design the PCR system. It first identified key components required, including third-party non-deliverables. Then, it discussed the behaviour of the system and its users throughout a code review exercise. Subsequently, it presented the initial prototype using wireframe diagrams. Furthermore, it explored the design of the Backend API endpoints. Lastly, it modelled the data stored in the database via an entity-relationship diagram.

In the chapter that follows, the implementation of the PCR system is discussed.

5 Implementation

The implementation of the PCR system was the most involved part of the project. This chapter begins by outlining key Software Engineering practices that were followed during the implementation phase. Then, it carefully selects technologies for each component of the system. Lastly, it shows several key features of the finished system and outlines how they were implemented.

5.1 Software Engineering Practices

5.1.1 Development methodology

Waterfall and Agile are two software development methodologies considered for the implementation of the PCR system. Every software project is unique. Both Waterfall and Agile approaches have several benefits and drawbacks that developers must consider (Bowes 2014). Waterfall follows several fixed phases: problem definition, requirements analysis, design, implementation, testing, deployment, and maintenance. In contrast, Agile practices treat software projects as constantly evolving, attempting to account for the complexity, unpredictability and evolutionary nature of software. Agile focuses on continuous delivery to a customer or end-user.

For the development of the PCR system, a Waterfall approach is a better fit for the following reasons:

- The system has a set of well-defined requirements.
- The project has a fixed timeline.
- Development does not require customer involvement.
- Development will be undertaken by a single developer and most Agile practices such as pair programming and code reviews require at least two developers.

However, the waterfall approach defines implementation and testing as two separate phases where testing occurs after implementation. This approach was modified for Backend implementation to incorporate Test Driven Development (TDD). TDD is a software development process in which developers only write code to pass a failing test (Beck 2015). Firstly, the developer writes an initially failing automated test case, and then writes the minimum amount of code required to pass the test. Upon completion, the developer refactors the code to enhance performance and improve quality while ensuring the test case passes. Developing the Backend in this fashion verifies that the REST API returns correct responses, ensuring a smoother Frontend development experience as the Backend is thoroughly tested.

Furthermore, in the waterfall approach, integration and deployment are carried out after implementation. However, this approach was modified for implementation the PCR system in order to allow the inclusion of continuous integration and delivery (CI/CD). CI/CD is a DevOps technique that simplifies development and speeds up deployment. It handles challenges surrounding integration and deployment throughout implementation rather than at the end of the project (Wallen 2019). This technique ensures minimal unexpected deployment delays and reduces uncertainty due to the waterfall approach.

In summary, the development of the PCR system follows a modified Waterfall approach that incorporates TDD and CI/CD.

5.1.2 Version control and repository management

In addition to a development methodology, utilising version control and repository management software is key in Software Engineering. The implementation of the PCR system utilises Git version control and GitLab to manage the repository.

Version control is software for managing and tracking changes to software code, scripts, and documentation. Git version control provides a repository for the PCR system. The repository contains all the Frontend and Backend code, along with scripts and code-related documentation. It maintains a history of every file, allows for rollbacks, and provides branching and merging capabilities. In addition, the GitLab repository manager manages the Git repository and provides functionality to create pull requests, compare versions, and merge branches. The implementation of the PCR system utilised a feature branching strategy on GitLab. This involves developing features in a dedicated branch instead of the main branch, then merging the feature branch into the main branch Atlassian (2021). Such an approach, therefore, enables continuous integration and delivery.

5.1.3 Continuous integration and delivery

The development of the PCR system employs both continuous integration and delivery. On one hand, continuous integration (CI) ensures changes to the repository are verified by an automated test and build process. It significantly reduces integration problems as errors are detected quickly (Fowler 2006). On the other hand, continuous delivery (CD) automatically deploys approved changes into a staging environment. The PCR system's Git repository is set up so that when code is committed and pushed to GitLab, a pipeline runs automated tests. The pipeline then deploys changes to a staging environment accessible outside the development environment, helpful for testing functionality on mobile and tablet devices.

5.1.4 Clean code

As discussed previously, a Test-Driven Development approach is taken to implement the Backend REST API. This approach ensures test suites evolve throughout implementation, and in the end, serve as helpful documentation for future development. When a developer expresses doubt or uncertainty about the purpose of a class or function, its test class provides examples of expected behaviour.

To complement this approach, the implementation of the PCR system aims to produce software using clean and expressive code that is easy to read and write, with few comments. Poorly written code can result in significant waste of project resources. Although clean code is not a Software Engineering principle, it is a development style that leads to modular and maintainable code (Martin 2008).

5.2 Choice of Technologies

As identified in the previous chapter via a system architecture diagram, the PCR system has several key components. However, programming languages, development frameworks, and key libraries must be carefully selected before implementation. In this section, the choice of technology for each component is made and justified.

5.2.1 Frontend

Section 4.3 presented the initial prototype of the PCR system. The prototype reveals the system reusing user interface components repeatedly. When choosing a Frontend development

framework, there are numerous options available. However, as the Frontend is only required to communicate with the Backend REST API, and all frameworks can achieve this, the chosen framework should focus on reusing components as much as possible.

React is a flexible, efficient, and fast JavaScript library for user interface (UI) development. React leverages resuability heavily. Complex interfaces developed in React are composed of reusable UI components. These components are small and isolated pieces of code that utilise state to remember things.

In addition, React has a variety of well documented open-source design libraries, such as Material UI and Elastic UI. Design libraries are excellent for creating interactive applications that are aesthetically pleasing and fully functioning on all devices, including mobile. A design library consists of hundreds of pre-designed components, such as text boxes, tables, and navigation bars.

React, and Elastic UI were used as the JavaScript framework and design library. Furthermore, many JavaScript libraries are available in React, such as Axios, to handle requests and responses from the Backend and Moment to handle and format dates and times.

Moreover, to develop the Frontend of the PCR system in JavaScript, several other technologies are necessary. Npm is a package manager used to manage and install modules and dependencies in JavaScript. ESLint is a static analyser used to maintain code quality against a user-defined standard. Finally, Create React App is a tool for creating a blank React application, with default configuration for Babel, Webpack, and more.

5.2.2 Backend

Many Backend development frameworks are available in various programming languages. Each framework has various built-in functionalities and libraries available. Comparing and selecting the correct framework is crucial as the Backend must communicate with third-party components, such as the email server and the message broker.

The Django Web Framework (Django 2021) and the Spring Framework (Spring 2021) are two popular backend technologies. Both have many modules offering various services and plugins. Django and Spring both provide built-in authentication, security features and are highly customisable. In addition, both frameworks provide support for all third-party components required.

However, Django has extensive built-in support for the data model. Django models, represented in Python, allow the definition of database tables, fields, and their constraints in code. These model classes can be extended to encapsulate data interaction logic outlined in the data model design in Section 4.5. Using migrations, Django automatically propagates changes to the external database. Its Object-Relational Mapper provides an interface to the database through the defined models. The Django REST Framework allows the creation of RESTful APIs, providing boilerplate code for serialisation, authentication and routing, and support for testing API endpoints. Moreover, Django also has an automatic customisable admin site, which can allow administrators to add instructors to the PCR system.

Overall, the PCR system has many functional and non-functional requirements but limited resources. Both frameworks provide support for all the third-party components required. Therefore, the Django Web Framework is the better-suited technology for the Backend as it provides built-in REST functionality, the data model, and the admin site.

In addition, to develop the Backend of the PCR system in Python, several other technologies are necessary. Pip is a package manager used to install libraries and dependencies in Python. Pipenv provides a virtual environment to manage dependencies using a Pipfile. Flake8 is a static analyser used to ensure code conforms to the PEP8 style guide.

5.2.3 Non-deliverable components

The chosen Backend development framework, Django, supports five relational databases, including PostgreSQL and MySQL. It provides a straightforward way of configuring the database to work with the Backend. A PostgreSQL database was chosen and configured during implementation as it was readily available on the deployment server.

Celery manages worker nodes. It receives tasks from a message broker, assigns them to one or more worker nodes, which then execute the task. The workers can make changes in the database and send emails. Redis is a message broker; it accepts and forwards tasks. The Backend can send long-lasting tasks to Redis, which are subsequently executed by Celery worker nodes.

Django supports integration with Celery and Redis. Additionally, Django supports any SMTP email server, such as Twilio SendGrid or Gmail.

5.3 Key Features and Implementation Details

Thus far, the implementation chapter has outlined the Software Engineering practices followed and justified the choice of technologies. This section covers the key features of the finished PCR system. It briefly outlines how they were implemented and what libraries were used. It is important to note that all design elements are from the Elastic UI design library (Elastic 2021).

5.3.1 Data model

Django's Object-Relational Mapper provides an interface to the external PostgreSQL database through Django models. Django models are represented as classes in Python. For example, the Initial Submission database table contains three attributes, a student ID foreign key, an assignment ID foreign key, and a base64 file for the student's code. Listing 5.1 shows the representation of the initial submission table as a model class in Python. The class extends the Django *models.Model* class to inherit numerous functions, including the *save* function, which is the interface Django provides to save models to the database. The model shown contains a base64 file field, represented as a *TextField*. It also contains two *ForeignKey* fields that are related to the Student and Assignment model, to represent the student ID and assignment ID foreign keys.

The data model is implemented by extending and overriding the *save(self, *args, **kwargs)* function in each model. In the initial submission model presented below, overriding the save function ensures that it is impossible to save a submission if the submission deadline has passed or if the assignment has not been released. This is implemented by utilising the *can_submit_initial_code* property on the assignment object. Listing 5.2 presents the *can_submit_initial_code* property function that belongs to the Assignment model class. The function first retrieves the current date and time using the *datetime* Python library. It then compares it with the assignment release date and submission deadline. The function returns true only if the assignment submission stage is open.

In summary, the data model is implemented by overriding the *save* function in each model class. This is done to ensure the integrity of the underlying data.

```
class InitialSubmission(models.Model):
   student = models.ForeignKey(to=Student, on_delete=models.CASCADE, blank=False,
       null=False)
   assignment = models.ForeignKey(to=Assignment, on_delete=models.CASCADE,
       blank=False, null=False)
   base_64_file = models.TextField(blank=False)

   def save(self, *args, **kwargs):
      if not self.assignment.can_submit_initial_code:
         raise ValidationError("Assignment is not accepting initial submissions")
      return super(InitialSubmission, self).save(*args, **kwargs)
```

Listing 5.1: *The initial submission database table implemented as a Django model class in Python. It shows three attributes: student ID foreign key, assignment ID foreign key, and a base64 file, represented using Django's TextField and ForeignKey fields. The class overrides the save function to prevent an initial submission from being saved if the deadline has passed or the assignment has not been released.*

```
@property
def can_submit_initial_code(self):
   now = datetime.now()
   released = now >= self.release_date
   overdue = now >= self.submission_deadline
   return released and not overdue
```

Listing 5.2: *The can_submit_initial_code property function in the Assignment model class. It shows how the function computes the current date and time and compares it with the assignment release date and submission deadline to return true only if the assignment submission stage is open.*

5.3.2 Creating an assignment

Creating an assignment is a complex step. It requires an instructor to enter information for each field, up to a maximum of 18 different fields. The initial prototype attempted to retrieve all this data from the same page. The wireframe diagram, shown in Appendix D.4, shows too many input fields on one page.

As wireframes were used to influence implementation, the page was redesigned. The new design simplifies the process of creating an assignment by breaking the process into five steps. At each step, there are buttons to progress to the next step or return to the previous step. Attempting to progress to the next step without completing all the required fields will notify the instructor through an alert and not allow them to proceed. In React JS, creating an assignment is a complex page requiring many hundreds of lines of code and countless complex components, including date-time pickers.

Step 1, as illustrated in Figure 5.1, requires the instructor to enter the title, weight and task description and select the rubric the new assignment will use. The task description can be formatted using Markdown. In the Frontend, a Markdown Editor component from the Elastic UI design library provides the editor itself. In the Backend, a django-markdownfield is used to store Markdown-formatted text in the Assignment model class. It ensures the Backend can validate the formatted task description and return an appropriate error message if the format is invalid. Furthermore, as with all steps, this step provides helpful text under each input field to guide

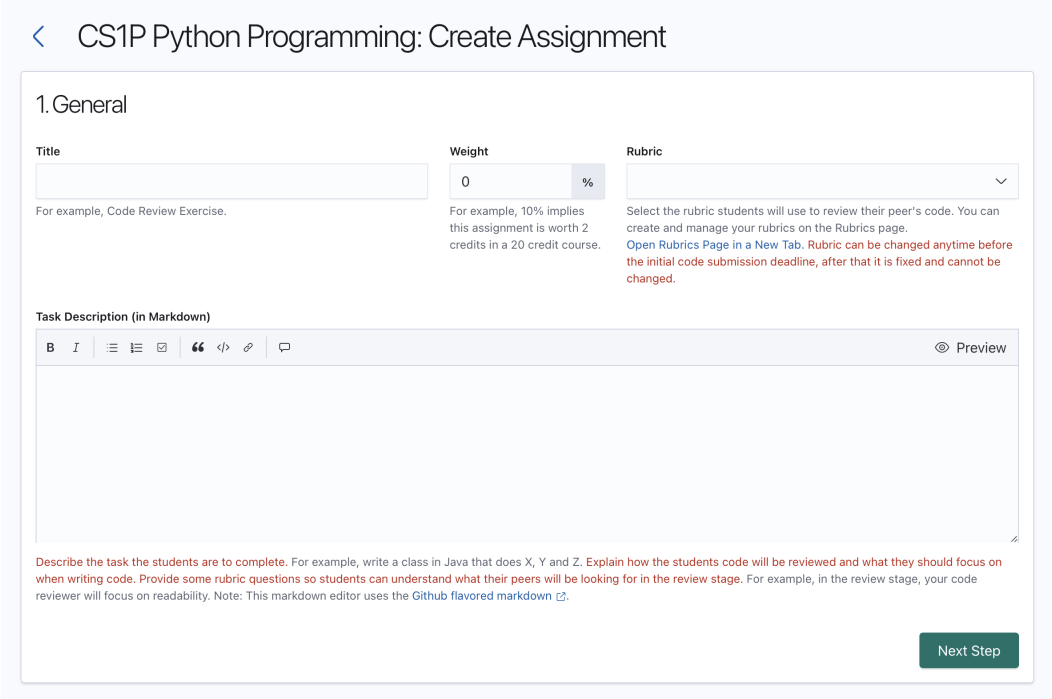


Figure 5.1: *The create new assignment page on step 1 of 5 allows an instructor to set the assignment title, weight, rubric and task description. It provides instructors with a Markdown editor, which allows formatting of the task description.*

the instructor to create a new assignment. For example, the task description field explains what details an instructor must enter, "describe the task the students are to complete", and "explain how the student's code will be reviewed and what they should focus on when writing code".

Step 2, as shown in Appendix I.2, requires the instructor to choose the structure of the assignment; this cna be either formative or summative. It explains the different stages of the peer code review exercise and outlines the difference between formative and summative assignments. It then informs the instructor of consequences related to students not submitting their code. Step 2 also influences future steps as summative assignments have an additional deadline and require a final submission weight. In React, the selected assignment structure is stored in the Create Assignment component's state. Future steps retrieve the state, then display fields the instructor needs, based on the assignment's structure.

Step 3, as illustrated in Figure 5.2, requires the instructor to set the release date and deadlines for each stage of the assignment. As the assignment structure in the figure is formative, there is no option to set a deadline for the final code submission. Similar to other steps, this step provides sensible and helpful defaults that are easy to change. For example, it will automatically space all deadlines one week apart, releasing the assignment on the following Monday.

Step 4, as shown in Appendix I.4, requires the instructor to assign weights to the assignment for grading. In total, weights must sum to 100%. A check at the bottom of the page ensures this. If weights do not sum to 100%, the page will not allow the instructor to proceed to the next step. Under each weight, step 4 helps instructors to understand the different weights and provides guidance. For example, for review participation, "Students must fully complete and answer all questions (from the rubric) in all three reviews to get the marks".

Step 5, as shown in Appendix I.5, provides a valuable overview of the first four steps. It includes

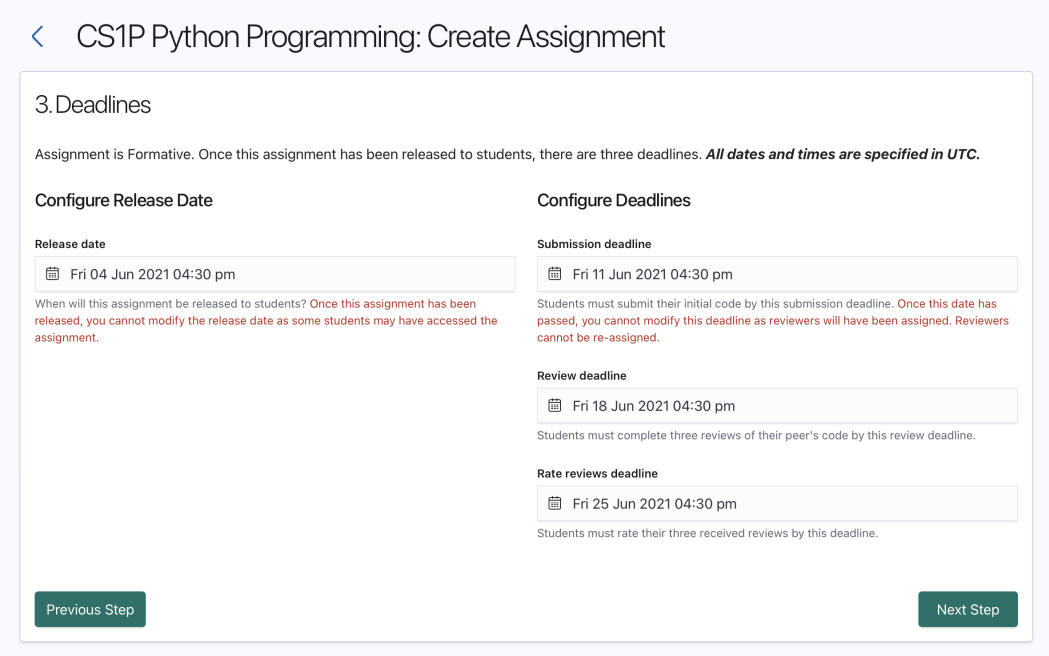


Figure 5.2: *The create new assignment page on step 3 of 5 displaying several date and time fields which allow an instructor to set the assignment release date and configure deadlines for each stage of the peer review assignment.*

everything the instructor has entered in a read-only format, minimising the potential for instructor mistakes. For example, it includes, "Students must submit their initial code by Monday 28th June at 4:30 pm for 30% of the marks.". This is possible in React as the Create Assignment component is a parent component that manages its own state, which includes everything the instructor has entered in the first four steps. Lastly, in step 5, the instructor can return to the previous steps or proceed to create the assignment, which sends an HTTP POST request to the Backend REST API with data from the Create Assignment component's state. As the component knows the assignment structure, it sends only the data required by the endpoint. Once React receives a response, if it is successful, the assignment is created. If unsuccessful, an error message is displayed. However, the Frontend ensures that all data is inputted and formatted correctly, minimising the possibility of Backend returning an unsuccessful response.

5.3.3 Completing a peer code review exercise

A formative peer code review exercise consists of three stages: initial submission, peer review, and rate peer reviews. The assignment has a release date and a task description. Each stage has weights and deadlines. In the Frontend, the Moment library was used to support time zones to ensure all students have the same deadline.

Figure 5.3 illustrates a peer code review exercise on the student assignment page. It shows the assignment format and task description. Then, there are three collapsible sections, one for each stage of the assignment. In the Frontend, an Accordion component from the Elastic UI design library provides the collapsible sections for each stage, allowing users to show and hide stages as they wish. Note that summative assignments show a fourth stage for final code submission. Alongside each stage are its weight and deadline. The weight is represented as a percentage of the assignment. The deadline is represented as a live countdown, for example, "Due in 1 day 22 hours". The react-countdown library was used to implement the deadline as a countdown that

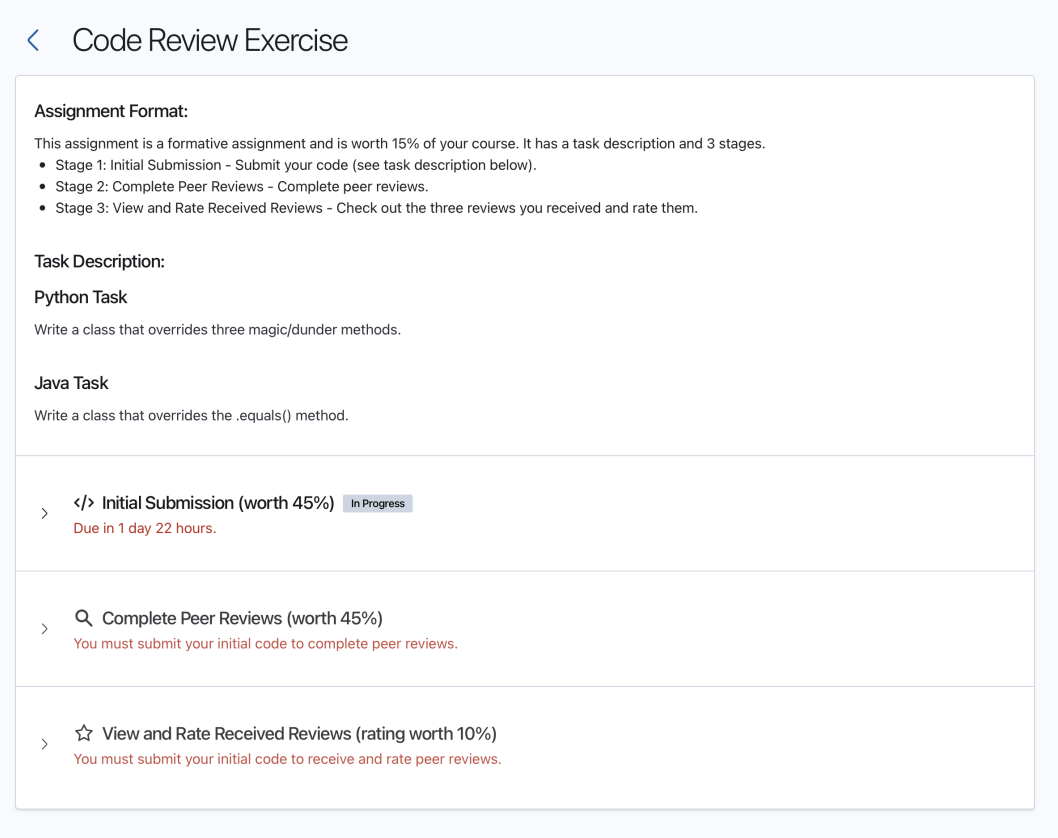


Figure 5.3: *The student assignment page showing a peer code review exercise underway. It outlines the assignment format and presents the task description. Then, it shows a section for each stage of the assignment. Clicking on a stage reveals further information.*

does not require a page refresh. Clicking on a stage reveals further information.

Appendix I.6 shows the Initial Submission stage open. The stage first informs the student how many marks are available for both submitting code and code quality. Then, it provides a submission area where the student can enter their code and click save to upload. The submission can be modified anytime before the deadline. Both the initial and final submission stages use the same underlying UI Component with different Backend API endpoints, utilising reusability in React. The Backend stores the code encoded in base64. In React, when a student submits their code, the js-base64 library encodes the code using base64 encoding. This library is used to ensure student code is correctly converted to base64. It correctly encodes single quotes, double quotes, and numerous other characters that the built-in JavaScript base64 encoder has trouble handling.

Appendix I.7 shows the peer review page. When a student opens the peer review stage on the assignment page, the three reviews they must complete are listed. Clicking on a review opens it on the peer review page. This page first shows the code the student must review and then informs them that the process ensures anonymity. It then provides information on what makes a good review and how the review will be rated. Finally, it allows the student to review the code using the rubric questions, and make changes to the review anytime before the deadline.

Figure 5.4 presents the rate review page. When a student opens the rate review stage on the assignment page, the three reviews they received and must rate are listed. Clicking on a review opens it on the rate review page. The page first shows the student's code and the review they

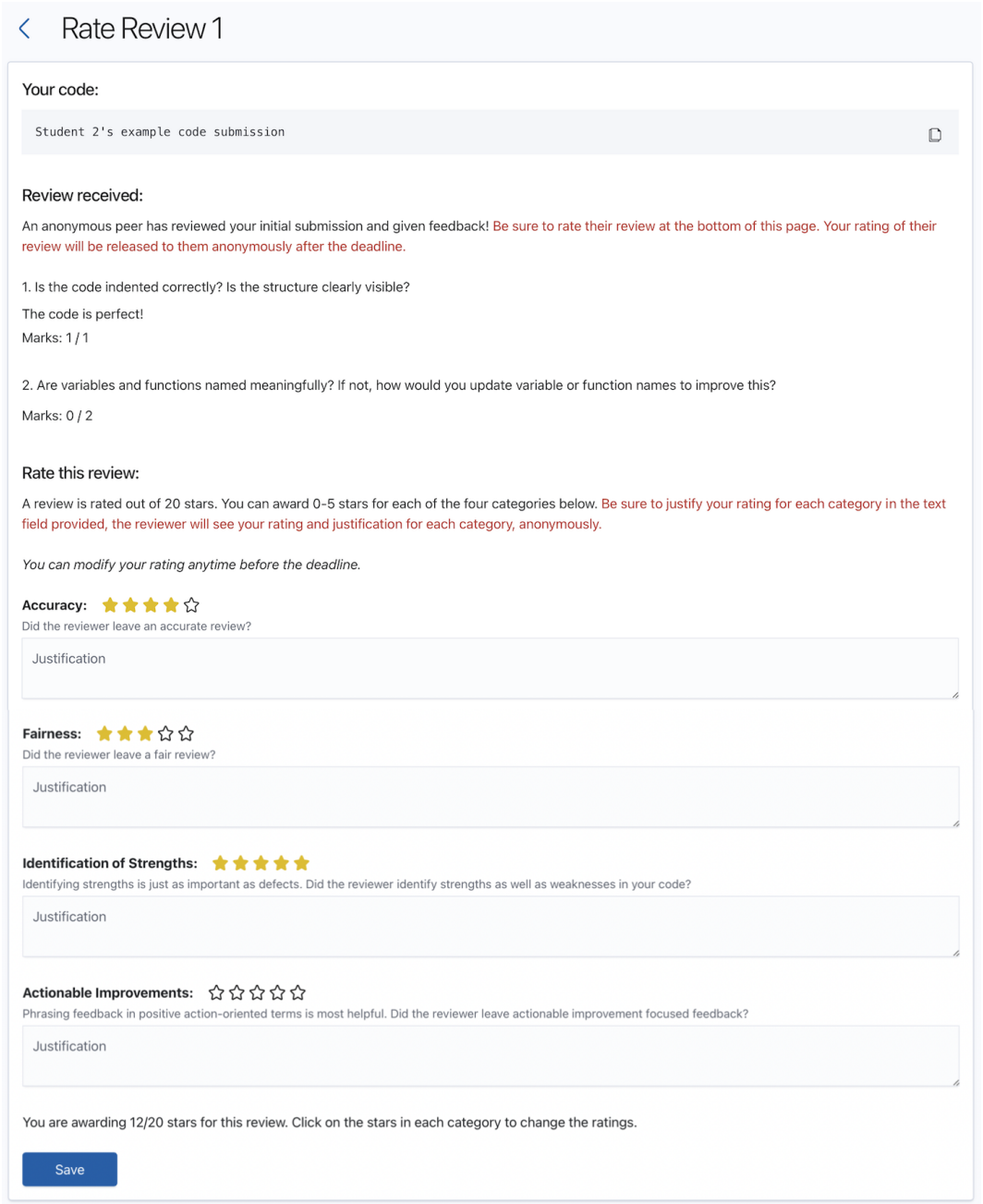


Figure 5.4: *The student rate review page showing a student's code, a review they received, and four categories a student must use to rate the review. For each category, it presents a 0-5 star-based rating and a text field for students to justify their reasoning.*

received from their peer. It then informs the student that a review is rated out of 20 stars, and they can award 0-5 stars for each of the four categories. It then allows the student to award stars and justify their reasoning in an open-ended manner. The student must justify each category to complete and save the rating. For example, ii the figure, the student is awarding four out of five stars for review accuracy.

5.3.4 Badges

With regard to badges, students are awarded blue badges for completing reviews, and bronze, silver, and gold badges for high-quality reviews. When a student reviews their peer's code, that peer rates the student's review. The rating is completed out of 20 stars based on four categories: accuracy, fairness, identification of strengths, and actionable improvements. The system awards the student a bronze badge for completing 14-star reviews, a silver badge for completing 16-star reviews, and a gold badge for completing 18-star reviews.

Badges are awarded in increasing increments. For example, if a student completes seven 18-star reviews, they are awarded gold badges for completing one 18-star review, three 18-star reviews, and five 18-star reviews. The next gold badge the student will be awarded is for ten 18-star reviews.

Appendix I.10 shows the student badges page. It shows seven badges a student has been awarded, including one gold badge for completing an 18-star review. As seen in the figure, students are not awarded a badge for every review. The review completion badges the student has are for completing one review, three reviews, five reviews, and so on. The next review completion badge the student will be awarded is for 25 reviews. Badges that have not been earned by the student appear fainter in the user interface.

The implementation of badges is solely done on the Frontend. The Backend REST API and the database do not contain any badge-related logic or data. The badges page loads the rating of every review the student has completed. The number of reviews it returns is the number of reviews the student has completed. The React component computes and presents badges the student has been awarded using the data received from the API call.

The badges are Scalable Vector Graphics (SVG) images that were designed from scratch using basic shapes. SVG images are pure XML, meaning they load incredibly fast. More importantly, this implementation allows the JavaScript Frontend to write on top of a badge SVG. This is how the system avoids storing badge data in the database.

5.3.5 Automatic reviewer designation after code submission deadline

In order to automatically assign reviewers after an assignment's code submission deadline, task delegation is required. The Backend performs task delegation using Redis and Celery. Redis is set up with its default configuration and runs on localhost port 6379. Celery is configured and integrated with Django (Celery 2021). Starting a Celery server is done by specifying the location of the celery integration files in the Backend within the virtual environment. The Celery server spawns child processes to execute tasks it receives from Redis and can utilise several Backend functionalities, such as sending emails and accessing the database.

With task delegation set up, the Backend needs to schedule tasks. Celery Beat is used to send a pre-defined task to Redis at regular intervals. Celery child processes execute the task in the background.

Listing 5.3 shows a Python function decorated with *@shared_task()*, representing the task itself. It retrieves all assignments that need reviewers assigned, ordered by the submission deadline. It does this by querying the database using Django's Object-Relational Mapper. If the query returns assignments, it chooses the first assignment to assign reviewers for as its submission

deadline is the earliest. Finally, to assign reviewers to the found assignment, it uses the function *assign_reviewers_to_assignment*.

With the task defined, the Backend sets the Celery Beat schedule. Listing 5.4 shows the task name and its execution schedule, every 30 seconds. At each interval, the task is sent to Redis and, shortly after, executed by a Celery child process. Each time the task executes, and there is an assignment that needs reviewers, it will assign them.

```
@shared_task()
def assign_reviewers_task():
   now = datetime.now()
   assignments = Assignment.objects.filter(
      have_reviewers_been_assigned=False,
      submission_deadline__lte=now,
      review_deadline__gte=now
   ).order_by('submission_deadline')
   num_assignments = assignments.count()
   if num_assignments > 0:
      assignment = assignments[0]
      assign_reviewers_to_assignment(assignment)
```

Listing 5.3: *A Python function, which shows the code executed by the scheduler every 30 seconds to find assignments that need reviewers assigned. When an assignment is found, assigning reviewers is delegated to the assign_reviewers_to_assignment function.*

```
CELERY_BEAT_SCHEDULE = {
   "scheduled_task": {
      "task": "api.tasks.assign_reviewers_task",
      "schedule": 30.0,
   },
}
```

Listing 5.4: *A Python variable, which configures the Celery Beat schedule to execute the assign_reviewers_task shown in Listing 5.3 every 30 seconds.*

5.4 Finished System

A 12-minute video demonstration of several features of the finished PCR system is on YouTube. The URL to access the demo is https://youtube.com/watch?v=lAHlKMrK0iA.

A 22 page user guide, titled *User Guide.pdf* containing step-by-step guidance on using the system and all screenshots of the finished PCR system is in the compressed folder.

In addition, the URL to try out the application is https://peer-code-review-system.herokuapp.com. Access details for an instructor and four student accounts are in Appendix J.

Alternatively, to build and run the PCR system locally, source code files are in the *src* folder, inside the compressed folder. Instructions on how to build, set up the environment, and install required dependencies are in *src/readme.md*.

5.5 Chapter Summary

This chapter presented the implementation of the PCR system. It first outlined key Software Engineering practices followed, including a modified Waterfall approach, Test-Driven Development, Continuous Integration, and Continuous Delivery. It then selected technologies for each component of the system and provided the rationale behind the decision. Subsequently, it presented several key features of the PCR system and outlined how they were implemented. However, it is important to note that other key complex features are not presented in this chapter. These include sending invitation emails to students when added to a course for the first time, the computation and download of student grades as a CSV file, and authorisation-based React routing, which ensures students cannot access instructor functionality, and vice-versa.

In the chapter that follows, the evaluation of the PCR system is presented through automated and manual testing, and a user evaluation.

6 Evaluation

This chapter presents the evaluation of the PCR system in three parts. Firstly, automated unit and API tests ensure the expected behaviour of the Backend REST API. Subsequently, manual acceptance tests ensure the requirements outlined in Chapter 3 are met. Lastly, a user evaluation of the system to determine system usability, the effect of rating reviews on review engagement, and the effect of badges on student motivation.

6.1 Automated Testing

As explained in the previous chapter, automated testing of the Backend was carried out continuously throughout implementation via Test-Driven Development. Django's TestCase and the Django Rest Framework's APITestCase were used to test the correctness of the Backend via unit and API tests. They both extend from the Python standard library module unittest. As far as unit testing is concerned, it aims to verify a small piece of code or a function behaves correctly as a self-contained unit. In terms of API testing, it aims to verify and demonstrate the functionality of the API endpoints.

During implementation of the system, several test suites evolved in the Backend. In total, the PCR system comprises 128 passing unit tests and 74 passing API endpoint tests. However, it is important to note that the automated testing approach caught many defects early on in the implementation of the Backend and saved precious project resources.

Tests follow the Arrange-Act-Assert (AAA) pattern. The AAA pattern is used to clearly separate what is being tested and how it is being tested from the setup itself (Beck 2015). Tests first arrange the objects and data required, then invoke a function, and check the result is as expected. The following unit and API test illustrate this:

Example unit test:

Test: test_student_id_must_have_length_7.

Arrange: Create several student objects with student IDs of varying lengths and one with a seven-digit student ID.

Act: Attempt to validate each student object against the student model class.

Assert: Expect validation to fail for all student objects except the one with a seven-digit student ID.

Example API test:

Test: test_only_the_course_instructor_can_edit_an_assignment.

Arrange: Create an assignment that belongs to an instructor. Then, sign in using a different instructor account.

Act: Attempt to edit the assignment using the REST endpoint.

Assert: Expect the REST endpoint to return HTTP 403 Forbidden and verify that the assignment has not been modified.

6.1.1 Code coverage

In addition to having automated tests, it is important to ensure that the tests cover a significant portion of the code base. Code coverage is a metric that measures how much of the code is covered by automated testing. Software projects aim to achieve around 80-90% coverage as exhaustive testing is impossible.

The Backend of the PCR system has code coverage of 87%. This does not mean that the system is free from defects or that the quality of the test cases is high, it simply gives confidence in that the automated tests exercise most of the system, and therefore, it is highly likely the system is functioning as expected.

6.2 Acceptance Testing

This section moves on to test whether the system meets the requirements listed in Chapter 3. Acceptance testing is a manual process that tests the system to determine if the system meets its requirements. Each acceptance test defines the expected behaviour of a feature and the actual result after being conducted.

Figure 6.1 shows the result of nine manual acceptance tests conducted on the PCR system. It describes each test and outlines the expected and actual result. A table that shows the result of all 52 acceptance tests can be found in Appendix K. It is important to note that several mutations of each acceptance test were conducted, for example, to verify whether an appropriate error message is displayed when a user enters invalid details or leaves a required field empty.

All non-negotiable, important, and desirable requirements outlined in Chapter 3 were implemented. Complete acceptance testing confirmed that the PCR system behaves as expected.

6.3 Pilot User Evaluation

Thus far, this chapter has presented automated and acceptance tests to ensure the PCR system functions and meets its requirements. Three students and two lecturers took part in a pilot evaluation to ensure the correct behaviour of the PCR system, provide feedback on system usability, and gather suggestions to improve system usability. The pilot aimed to get feedback and refine the system before conducting the user evaluation with a wider group.

In order to conduct the pilot evaluation, participants required credentials to access the system. Instructor accounts were created and sent to the lecturers via email. Students were asked for permission to be added to the system. Then, they were added to a course, which triggered the system to automatically send students a welcome email requesting them to set their password.

In the pilot evaluation, participants were asked to perform several different tasks via a survey. The survey contained different tasks for students and instructors. Instructors were asked to create a course, create a rubric containing questions, create a formative assignment, add a student to the course, download, and view the results from an existing assignment and more. Students were asked to browse through the course they were added to, complete three stages of a formative peer code review exercise, and view the badges they were awarded. Appendix L shows all the tasks for students and instructors.

Upon completing a task and before progressing to the next task, the survey asked participants three questions. The first two questions asked participants to rank how much they agree with the statements, "I thought completing this task was easy" and "Most people would know how to complete this task very quickly" on a scale of 1 to 5, from strongly disagree to strongly agree. It then asked participants the following open-ended question, "How could system usability be improved?".

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
SIGN IN	A student or instructor enters their email address and password and clicks Sign In.	The student or instructor is successfully authenticated and redirected to the instructor or student courses page.	As expected, passed.
INSTRUCTOR: CREATE COURSE	Click New Course, enter a title and number of credits and click Save.	A course is created and added to the instructor's list of courses on the courses page.	As expected, passed.
INSTRUCTOR: ADD A STUDENT TO A COURSE	Click Manage Students in a course, click Add Student, enter student id, given names, surname, and email address, and click Save.	A student is added to the course. If they are unknown to the system, a welcome email requesting them to set their password is sent.	As expected, passed.
INSTRUCTOR: VIEW RUBRICS	Navigate to the rubrics page.	The rubrics are listed. The instructor can click on a rubric to open and view details.	As expected, passed.
INSTRUCTOR: REMOVE QUESTIONS FROM THE RUBRIC	Open a rubric on the rubrics page, click Edit Rubric. Click the trash can icon next to a question. Then click save changes.	The question is removed from the rubric.	As expected, passed.
INSTRUCTOR: DOWNLOAD STUDENT FINAL MARKS WITH A BREAKDOWN OF RESULTS	Click Download Results for a completed assignment.	A CSV file is downloaded containing a breakdown of results for every student in the course for that assignment.	As expected, passed.
STUDENT: SUBMIT CODE BEFORE THE INITIAL SUBMISSION DEADLINE.	Open the assignment, click on Initial Submission to expand the section, enter code into the submission area and click Save.	The student's initial submission is uploaded/saved.	As expected, passed.
STUDENT: VIEW BADGES AWARDED	Navigate to the badges page.	The student can see all the badges they have been awarded for completing reviews and for high-quality reviews.	As expected, passed.
SYSTEM: TIME ZONE SUPPORT	Peers can partake in the assignment from different time zones. Update the time zone on the device and check to ensure deadlines are correct.	All students have the same deadline regardless of geographical location.	As expected, passed.

Figure 6.1: *The result of nine acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result. It shows user-specific acceptance tests for instructors and students.*

It is important to note that all three students were required to complete the pilot evaluation simultaneously as it involved taking part in a code review exercise. Each student reviewed two of their peer's submissions in the review stage.

Upon completion, suggestions for improvements from the survey were utilised and several features of the PCR system were refined to improve system usability. In addition, no defects were found in the system. The raw results captured from students and instructors can be found in the compressed folder, specifically in *compressed/data/raw/pilot*.

6.4 Final User Evaluation

6.4.1 Aims

The final user evaluation with a wider group is the most significant aspect of evaluating the PCR system. It aims to measure system usability, the effect of rating reviews on student engagement, and the effect of badges on student motivation.

6.4.2 Methodology

Before proceeding to examine the results of the evaluation, it is important to understand the methodology for evaluating the PCR system.

In the evaluation, participants are asked to complete a task-centric activity. Each task is accompanied with the question, "I thought completing this task was easy" and a 1-7 scale, from strongly disagree to strongly agree. This question is asked to understand whether participants find any particular task more challenging than another.

Furthermore, students and instructors are presented with a different set of tasks as they evaluate different functionalities of the PCR system. The system must be prepared for the evaluation to take place to ensure participants experience the entire system.

A course, "Test Course", and a summative assignment, "Test Assignment", are required to conduct the instructor evaluation. Two dummy students must complete the assignment to enable instructors to evaluate the usability of marking final student submissions and downloading and publishing results.

A course, "Final Evaluation", containing one assignment, is required to conduct the student evaluation. Appendix M shows the task description and the rubric of the assignment. A simple assignment task and two short rubric questions ensure students have enough time to submit their code and review up to three peer submissions.

The following two sub-sections list the evaluation tasks for instructors and students.

Instructor tasks: This sub-section contains five tasks instructors must complete in the user evaluation.

1. Sign in to the system and create a rubric containing two questions.
2. Create a course.
3. Create a dummy formative assignment for your course.
4. Add a student to your new course using the following credentials, 2324127 Mohammad Majid - use these details because the system will send an email to the student.
5. An existing course called "Test Course" has one assignment called "Test Assignment". It is a summative assignment and has been completed by two students. Complete the following three sub-tasks for "Test Assignment".
   5.1. Mark the student's final submissions.
   5.2. Download the results of the assignment.
   5.3. Publish the results of the assignment.

Student tasks: This sub-section contains four tasks students must complete in the user evaluation. Task three, completing a formative peer code review exercise, requires more than two students to complete the evaluation simultaneously. Therefore, it has implications on the timing of the student evaluation.

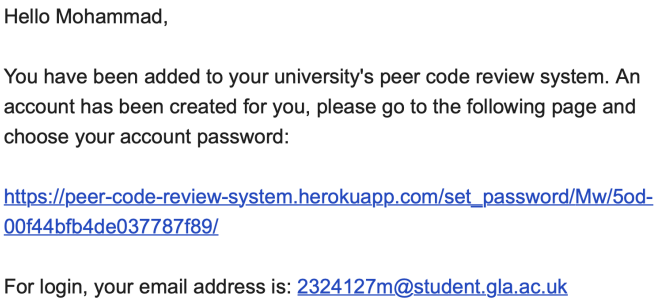
1. You have received a Welcome email asking you to join the peer code review system. Follow instructions in the email and sign into the system.
2. Browse through your courses and assignments.
3. Complete the assignment titled "Evaluation". This will involve 3 stages, submitting the initial code, completing up to 3 peer reviews, and rating any reviews you received.
4. Explore the badges page.

Timing and access details

Instructors are able to complete the evaluation at any time. However, this is not possible for students as at least two students are needed to complete a peer code review exercise. Therefore, to allow students to complete the peer review exercise, 10 time slots are needed over a period of two weeks. Students are asked to sign up to a time slot and complete the evaluation at that fixed time. If students are unable to make it or are late, they must be told to change their time slot. This ensures students have enough time to complete the tasks. Each time slot releases the code review exercise on the hour. Students have 15 minutes to submit their code, 20 minutes to submit up to 3 reviews, and 15 minutes to rate reviews they received.

Participants require credentials to access the system. Instructor accounts are created and credentials are sent to them via email. However, the system automatically creates student accounts when students are added to a course for the first time. The system sends students a welcome email, as shown in Figure 6.2, and requests them to set their password.

The first task in the student evaluation is to follow instructions in the welcome email and sign into the system. It is important to note that students must be added to the course with their permission and must receive the welcome email 10 minutes before their time slot. This ensures they access the system for the first time immediately before or during the evaluation to avoid bias.


Hello Mohammad,

You have been added to your university's peer code review system. An account has been created for you, please go to the following page and choose your account password:

https://peer-code-review-system.herokuapp.com/set_password/Mw/5od-00f44bfb4de037787f89/

For login, your email address is: 2324127m@student.gla.ac.uk

Figure 6.2: *An email received by a student when added to a course on the PCR system for the first time. It informs the student they have been added to the system and provides instructions on how to set their account password.*

Survey

After completing the task-centric evaluation activity, participants are asked to rate their experience with the system. The System Usability Scale (SUS) is used to understand the usability of the system quantitatively.

Brooke (1995) developed SUS for a reliable, low-cost means of measuring system usability. It is an industry-standard that has shown to differentiate between usable and unusable systems. It consists of ten questions, found in Appendix N, that alternate between positive and negative statements, ensuring participants read and think about statements carefully. Participants are required to rank each question from 1 to 7, from strongly disagree to strongly agree. However, for the student

evaluation, the first statement, "I think that I would like to use this system frequently", is not needed as students only use the system to complete assignments set by instructors. Therefore, the student questionnaire contains nine SUS statements, and the instructor questionnaire contains ten.

Subsequently, students are asked several additional quantitative questions with the same 1–7 scale as above:

1. I feel more engaged in the review process because I know my reviews will be rated.
2. I am more likely to learn what makes a good review because I know my reviews will be rated.
3. It does not make a positive difference to my motivation when I see I will be awarded badges for completing reviews.
4. I feel more motivated when I see I will be awarded bronze, silver or gold badges for high-quality reviews.
5. I would prefer not to see badges in the system.
6. I enjoyed reviewing other students' code.
7. I feel I have learned from the review I received.

Questions one and two aimed to evaluate the effect of rating reviews on student engagement with the review process. Questions three to five aimed to evaluate the effect of being awarded badges on student motivation. Questions six and seven aimed to understand the student's experience quantitatively.

Furthermore, both students and instructors are asked three open-ended questions to understand their experience with the system.

1. What did you like the most about the peer review system?
2. What did you dislike the most about the peer review system?
3. Any other comments?

6.4.3 Evaluation results

The final user evaluation was conducted following the methodology outlined above. 17 level three and four honours university students and five instructors took part. All students were studying Computer Science or Software Engineering. The five instructors include four university lecturers and one PhD student. The raw results captured from students and instructors can be found in the compressed folder, specifically in compressed/data/raw/final.

System usability

Calculating the System Usability Scale (SUS) score of the system consists of four steps (Betteridge 2021). Step one is to convert participant ratings into points. For questions that generate a positive response, $points = rating - 1$. For questions that generate a negative response, $points = 7 - rating$. Points range from 0 to six, with six being the most positive response. Step two is to sum the points to get the total points for each participant. Step three is to calculate the SUS score of each participant by scaling the total points out of 100. In the final step, the SUS score of the system is calculated as the average SUS score over all participants.

Table 6.1 shows the points calculated from the results of the SUS questions in the instructor evaluation. It shows the total points, and SUS score for each instructor. The average SUS score for instructors is 82. Similarly, Appendix O shows the results for students. The average SUS score for students is 92. A SUS score is not a percentage. The average SUS score is 68, and a score above 80 is considered to be in the top 10% of scores (Sauro 2011). Therefore, with an average score of 82 and 92 from instructors and students, the PCR ranks in the top 10% on the System Usability Scale.

Participant	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Total Points	SUS Score
Instructor 1	4	4	5	5	3	4	4	4	4	4	41	68.33
Instructor 2	5	5	5	6	4	5	4	6	5	6	51	85
Instructor 3	4	5	5	6	6	6	6	6	6	6	56	93.33
Instructor 4	5	6	6	6	5	5	5	6	5	6	55	91.67
Instructor 5	4	4	4	5	4	4	4	5	4	5	43	71.67

Table 6.1: *The points calculated from the results of the System Usability Scale questionnaire for instructors. It shows the total points, and SUS score for each instructor. The average SUS score is 82.*

After analysing the student open-ended questions, several themes were identified. Firstly, students overwhelmingly report that the system is easy to use. Comments include "very intuitive system", "easy to use", "simplistic, yet modern and elegant", "very professional and intuitive", "dark mode is also a good addition, easy on the eyes", and "pleasing minimalist design". Secondly, some students disliked the evaluation assignment itself, saying "the exercises were quite trivial", "the time delay between tasks", and "long wait". However, it is reassuring to see that students had sufficient time to complete the assignment during the evaluation. Thirdly, students identify small usability improvements, including a warning alert when attempting to leave the page accidentally and a redirect to the assignment page once a student review is complete. Lastly, students identify potential new features that can be incorporated to improve usability, including an auto-refresh feature on the assignment page, submission area syntax highlighting and the ability to upload source code files directly.

The open-ended instructor questions followed a similar pattern. Firstly, instructors found the system "easy to use", the create assignment page "very well presented", and navigation as "smooth". Secondly, instructors mention several improvements to the system. Two instructors found there to be a lot of reading when using the system. Although this is true, it is necessary to ensure instructors have all the information they need to make informed decisions. Nevertheless, to improve the system, the majority of the reading can be relocated to a dedicated guidance page on the system. One instructor mentioned user interface surrounding the download and publish results buttons "could look a little better". Lastly, an instructor expressed a concern about the "added burden of rating reviews for students". However, this may not be an issue as students did not raise this concern. Nevertheless, further research can be conducted to understand students' experiences with the rate review stage.

Overall, it is clear from both the SUS scores and qualitative feedback that the system is usable.

Rate reviews

Students were asked if they felt more engaged in the review process because they knew their reviews would be rated. 88% of student participants agreed with this statement, with 65% of those students strongly agreeing. Students were also asked if they were more likely to learn what makes a good review because they knew their reviews would be rated. Similar results were found. 88% of student participants agreed with the statement, with 53% of those students strongly agreeing. In both cases, 6% of student participants disagreed with the statement. An additional 6% of student participants neither agreed nor disagreed with the statement.

After analysing the open-ended questions, 12% of students mention rating reviews in a positive light. One student states, "Having our reviews rated was a good addition. Means you are more likely to spend more time writing the review and thinking critically about how to write a good review". Another student says, "great to know how well you performed in rating and how well is the other person satisfied".

Overall, it is clear that students were more engaged in the review process as they knew their reviews would be rated.

Badges

Students were asked if it made a positive difference to their motivation, knowing that they will be awarded badges for completing reviews. 70% of student participants agreed with this statement, with only 12% of students strongly agreeing. 18% of students disagreed with the statement. An additional 12% of student participants neither agreed nor disagreed with the statement.

Students were asked if they would prefer to see badges in the system. 88% of student participants agreed with this statement, with 47% of students strongly agreeing. 6% of students disagreed with the statement. An additional 6% of student participants neither agreed nor disagreed with the statement.

Students were also asked if they felt more motivated knowing that they would be awarded badges for completing high-quality reviews. 70% of student participants agreed with this statement, with 41% of students strongly agreeing. 24% of students disagreed with the statement. An additional 6% of student participants neither agreed nor disagreed with the statement.

After analysing the open-ended questions, 18% of students mention badges in a positive light or suggest implementation improvements. One student states, "very good system, easy to use and encourages fair and correct code reviews through the use of badges and code review reviews". Another student says, "I would like to get my badge for doing a code review right away", as the system awards badges after completing the assignment. Although this is a suggestion is referring to system improvement, it is encouraging since it reveals the student's eagerness to be awarded. Lastly, one student says, "it was not clear what had and had not been earned". This is because badges that have not been awarded appear fainter on the user interface than badges that have been awarded.

Overall, it is clear that students were more motivated as they knew they would be awarded badges.

6.4.4 Limitations

Two limitations of the user evaluation have been identified. However, prior to discussing limitations, it is important to note that the user evaluation was taken place during the Covid-19 pandemic, and restrictions made recruiting participants challenging.

Firstly, all students were level three and four honours university students. This is not reflective of the general student population as there were no level one or two students. Students in early computing courses are generally less experienced with peer review and may have found the system less usable. In future research, student participants from all levels of computing courses should partake in the user evaluation.

Secondly, students completed the evaluation in one of ten fixed time slots. Time slots ensured that at least two students completed the evaluation simultaneously so that students had code to review in the peer code review exercise. However, some time slots had only two or three students, and others had up to five. Students who completed the evaluation with one other student only received one code review and only rated one review. Students who completed the evaluation with at least three other students received three code reviews and rated three reviews. All students had 20 minutes to submit reviews and 15 minutes to rate reviews in the evaluation. Therefore, some students may have felt under pressure due to time constraints when completing the evaluation. As a result, they may have focused too much on the assignment itself instead of the evaluation. In future research, evaluation can be carried out with an entire class or groups of more than four students at a time.

6.5 Chapter Summary

This chapter offered an overview of how the PCR system was evaluated through three stages. It first ensured the expected behaviour of the Backend through the automated unit and API tests. Afterwards, it showcased how acceptance testing was performed using a few examples to ensure requirements had been met. Then, it presented the pilot user evaluation used to refine the system. Lastly, it presented the final user evaluation in detail and summarised its results.

The next and final chapter summarises this dissertation, provides a short discussion on future work and, lastly, concludes the dissertation with a final reflection.

7 Conclusion

This chapter briefly summarises this dissertation, discusses future work on the PCR system, both in terms of new features and research. It then concludes this dissertation with a final reflection.

7.1 Summary

In summary, peer review allows peers the opportunity to critique and provide feedback on each others' work. Peer review eases the administrative burden on staff in an educational setting and provides students with timely feedback on assignment submissions, which enables them to reflect and improve their work. Code review is an established practice in software development and is gaining considerable attention in higher education. However, in education, problems with low review quality and a lack of motivation result in some students missing out on a valuable learning experience. This dissertation guides the reader through the development of a Peer Code Review (PCR) System. The system aims to enable course instructors to create and conduct formative and summative peer code review exercises with students, and meet the unique requirements of higher education. In addition, it aims to improve student engagement with the review process and increase student motivation to achieve better learning. The PCR system achieves this by allowing students to rate the peer reviews they receive and by awarding badges to students for completing reviews and high-quality reviews.

This report began by discussing relevant background research and key features of related products to understand the requirements of the PCR system. Afterwards, it gathered and prioritised the functional and non-functional requirements of the PCR system. Subsequently, it took a top-down approach to design the system. It then discussed the implementation of the system along with several key features. Finally, it ended with an evaluation of the PCR system.

The evaluation of the PCR system concluded that the system is usable. In addition, it found that students were more engaged in the review process and more motivated overall.

7.2 Future Work

In the implementation phase of the PCR system, all initially planned non-negotiable, important and desirable features were implemented. Future development of the PCR system should focus on the improvements recommended by instructors and students in the final user evaluation as well as the several new features, including:

1. Additional assignment configuration options for instructors, including an option to choose how many reviews each student must complete and an option to disable the rate review stage. This would enable greater flexibility in configuring assignments.
2. When an assignment is underway, provide instructors with assignment-specific data, such as the number of students that submitted their code. This would provide instructors the data they need to make decisions, such as changing an assignment deadline.

3. When a student is enrolled in several courses with peer code review exercises, they will have three or four deadlines for each assignment. A new page for students to view a summary of upcoming deadlines would allow students to manage their deadlines.
4. When a student reviews code, they follow a rubric. Providing them with the ability to comment on any line of the code has the potential to further improve the review process.
5. Many programming assignments follow similar rubrics, so allowing instructors to create public rubrics will enable instructors to utilise existing rubrics when creating assignments. This has the potential to save a considerable amount of precious time.

In addition to improvements and new features, there is a potential for future research using the PCR system. This dissertation found that rating reviews improves student engagement with the review process. However, further research may entail determining whether rating reviews is improving the quality of reviews. This can be achieved, for example, in a study involving students in higher education, using a control group with the ratings feature disabled to establish a cause-and-effect relationship.

7.3 Final Reflection

This section concludes this dissertation with a final reflection on the project. The project involved many aspects of software development, exposed me to numerous new technologies, and allowed me to apply and develop key Software Engineering practices learned at the University of Glasgow. Reflecting on many challenges faced during the project, I learned a lot and have grown as a Software Engineer.

Firstly, I realised the importance of carefully researching technologies when designing software. The newly established PCR system consists of several components, including the Frontend and Backend. Ensuring that all components were compatible with each other was crucial to the success of the project.

Secondly, I discovered that as a code base gets larger, structural complexities arise and, as a result, a software system requires additional effort to maintain. The system has over 15,000 lines of code in Python and JavaScript and utilises numerous libraries and modules. Software refactoring must be employed to code and to the structure of the code base to ensure the project remains manageable.

Lastly, I learnt the importance of evaluating software to understand system usability and gather feedback. Feedback is pivotal to understanding what a software system needs. The PCR system was refined after receiving user feedback in the pilot evaluation.

A Ethics Checklist

School of Computing Science
University of Glasgow

Ethics checklist for 3rd year, 4th year, MSci, MRes, and taught MSc projects

This form is only applicable for projects that use other people ('participants') for the collection of information, typically in getting comments about a system or a system design, getting information about how a system could be used, or evaluating a working system.

If no other people have been involved in the collection of information, then you do not need to complete this form.

If your evaluation does not comply with any one or more of the points below, please submit an ethics approval form to the Department Ethics Committee.

If your evaluation does comply with all the points below, please sign this form and submit it with your project.

---

1. Participants were not exposed to any risks greater than those encountered in their normal working life.
   Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g. walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g. ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback

2. The experimental materials were paper-based, or comprised software running on standard hardware.
   Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and PDAs is considered non-standard.

3. All participants explicitly stated that they agreed to take part, and that their data could be used in the project.
   If the results of the evaluation are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant.

   Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

4. No incentives were offered to the participants.
   The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

Figure A.1: *A copy of the Ethics Checklist, Page 1.*

5. No information about the evaluation or materials was intentionally withheld from the participants.
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

6. No participant was under the age of 16.
   Parental consent is required for participants under the age of 16.

7. No participant has an impairment that may limit their understanding or communication.
   Additional consent is required for participants with impairments.

8. Neither I nor my supervisor is in a position of authority or influence over any of the participants.
   A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. All participants were informed that they could withdraw at any time.
   All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

10. All participants have been informed of my contact details.
    All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

11. The evaluation was discussed with all the participants at the end of the session, and all participants had the opportunity to ask questions.
    The student must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.

12. All the data collected from the participants is stored in an anonymous form.
    All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

Project title Developing a peer review system

Student's Name Mohammad Majid

Student's Registration Number 2324127

Student's Signature usman majid

Supervisor's Signature

Date 21/10/2020

Figure A.2: *A copy of the signed Ethics Checklist, Page 2.*

B Activity Diagram

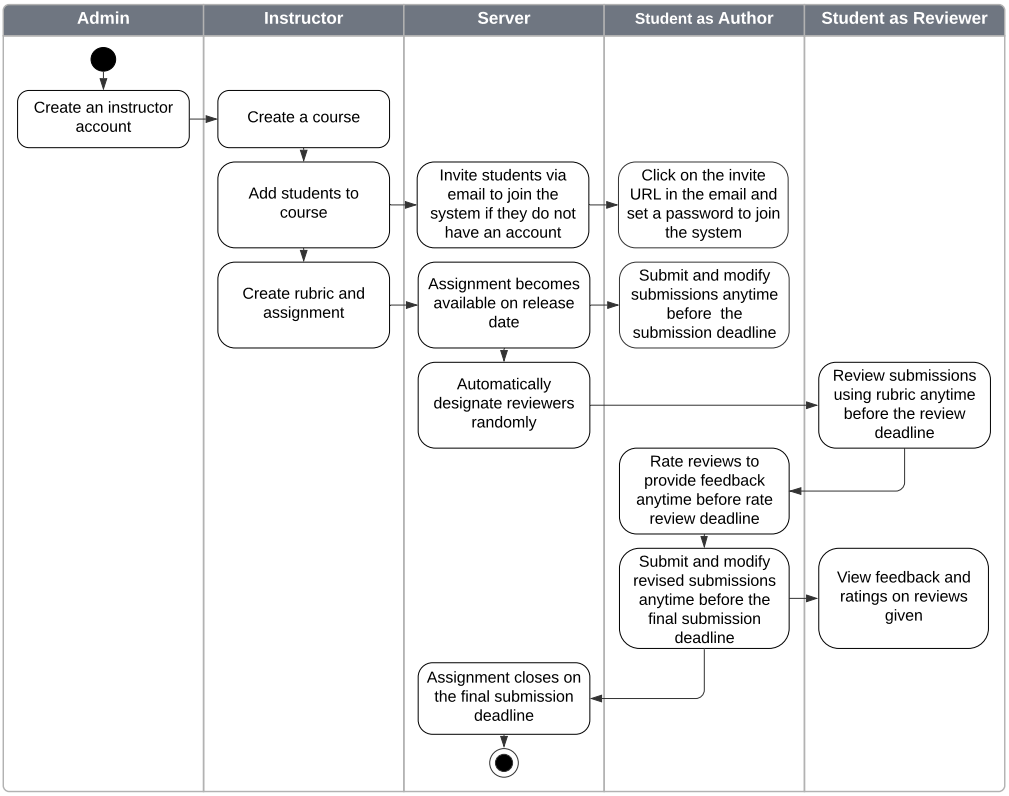


Figure B.1: *An activity diagram showing a new instructor joining the system and completing a peer code review activity with students. The diagram distinguishes between different user types and shows students rating the reviews they receive from their peers.*

C Authentication Wireframe Diagrams

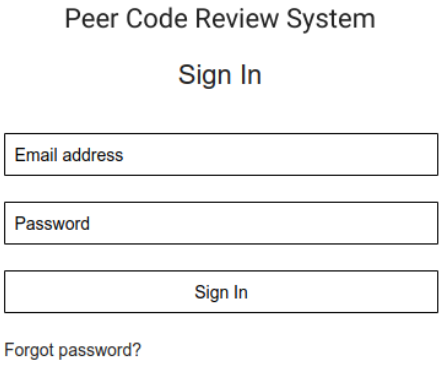


Figure C.1: *The design of the sign in page, depicted in a wireframe diagram. On the page, a user can enter their email address and password to sign in.*

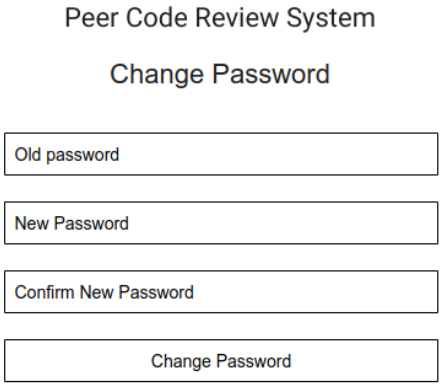


Figure C.2: *The design of the change password page, depicted in a wireframe diagram. On the page, a user can enter their old password, a new password, and the new password again to change their account password.*

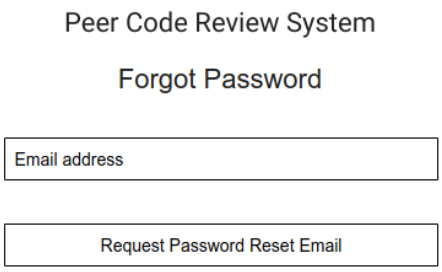


Figure C.3: *The design of the forgot password page, depicted in a wireframe diagram. On the page, a user can enter their email address to request a password reset email.*

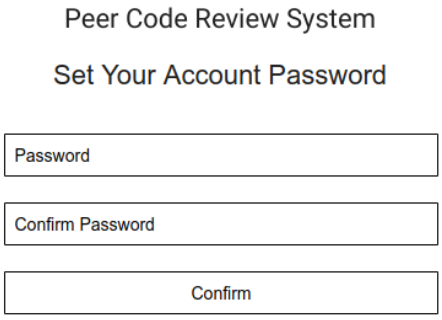


Figure C.4: *The design of the set password page, depicted in a wireframe diagram. A user lands on this page by clicking on the password reset link in the forgot password email. On the page, a user can enter their new password to change their account password.*

D Instructor Wireframe Diagrams

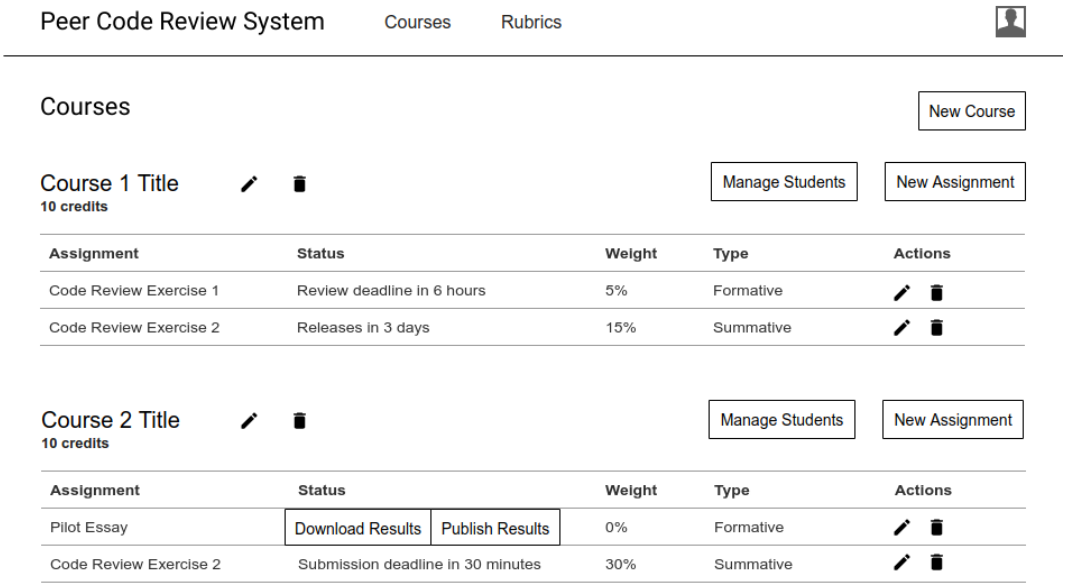


Figure D.1: *The design of the instructor courses page, depicted in a wireframe diagram. On the page, an instructor can create and manage all their courses and assignments.*

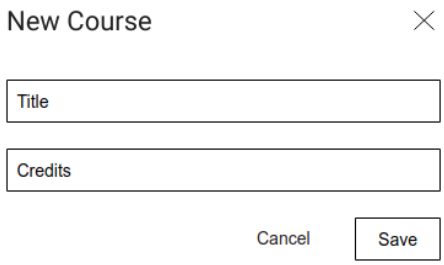


Figure D.2: *The design of the instructor new course modal, depicted in a wireframe diagram. This modal opens when an instructor clicks on New Course on the instructor courses page. It shows two fields required to create a course.*

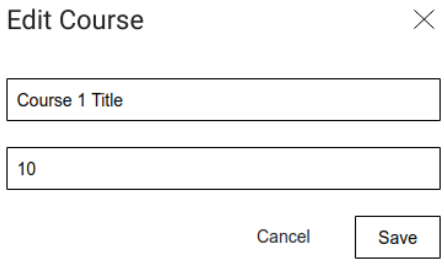


Figure D.3: *The design of the instructor edit course modal, depicted in a wireframe diagram. This modal opens when an instructor clicks on the edit pencil icon next to a course on the instructor courses page. It shows two pre-filled fields and allows the instructor to update the course title and number of credits.*

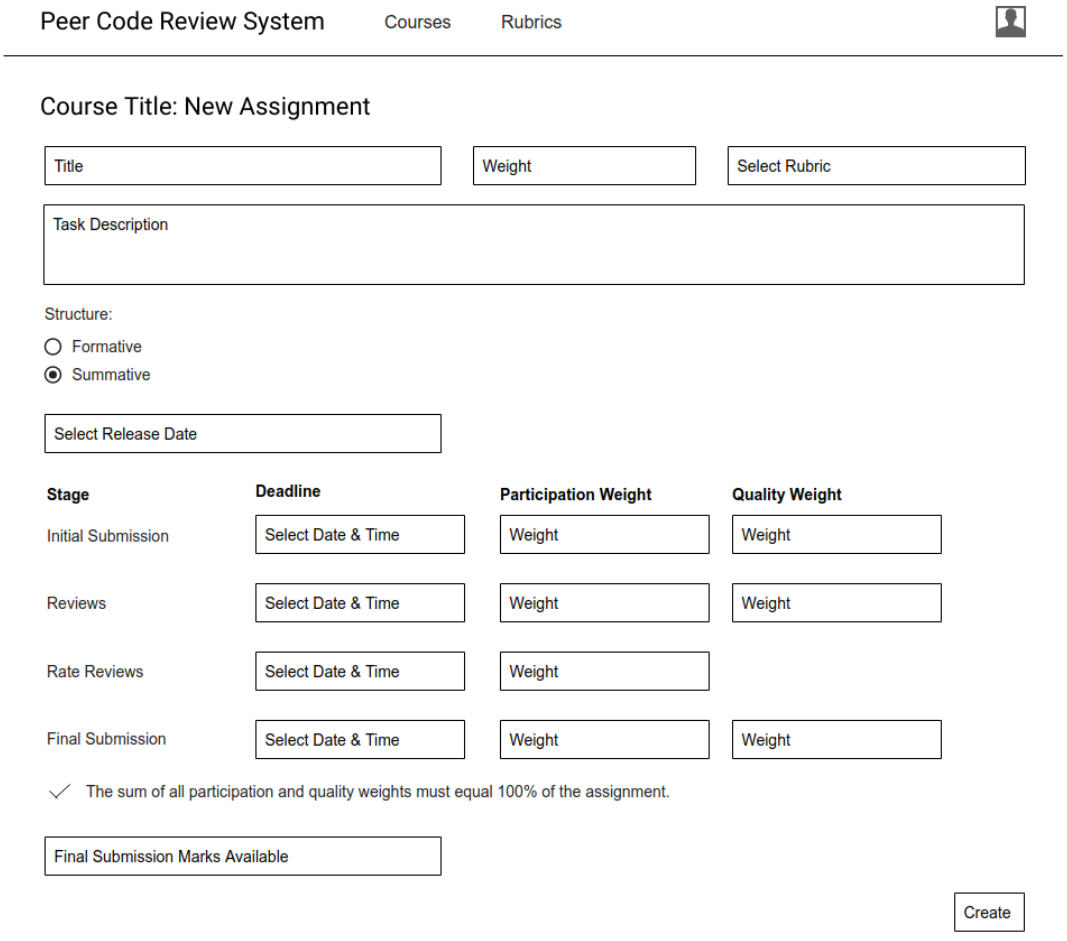


Figure D.4: *The design of the instructor create new assignment page, depicted in a wireframe diagram. It shows numerous different input fields required to create an assignment including weights and deadlines for all stages of the assignment.*

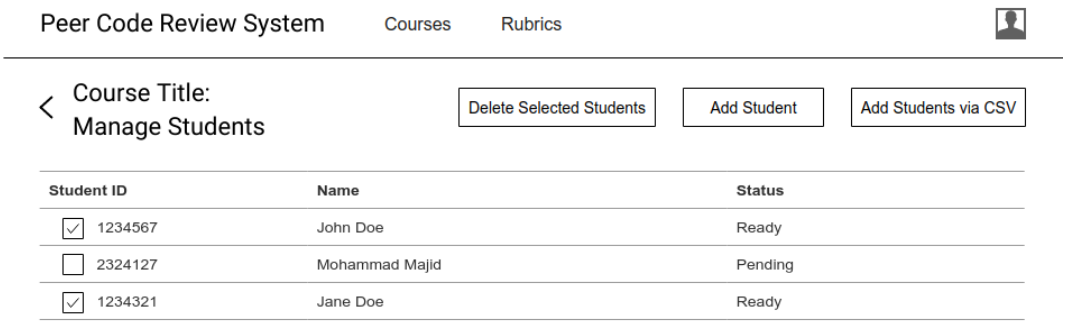


Figure D.5: *The design of the instructor manage students page, depicted in a wireframe diagram. It shows the students in a course and allows an instructor add and remove students.*

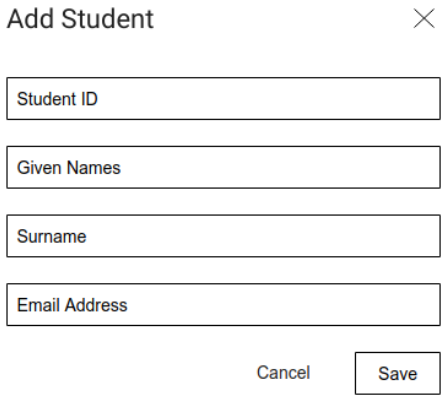


Figure D.6: *The design of the add student modal, depicted in a wireframe diagram. The modal opens when an instructor clicks on Add Student on the manage students page. It shows the four fields required to add a student to the system.*

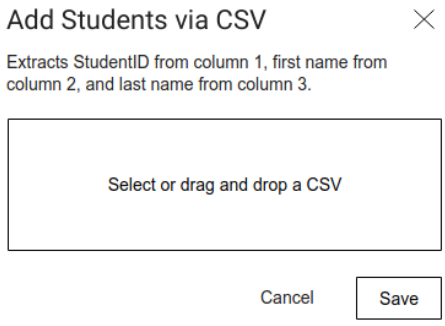


Figure D.7: *The design of the add students via CSV modal, depicted in a wireframe diagram. The modal opens when an instructor clicks on Add Students via CSV on the manage students page. It shows a single field that allows an instructor to upload a CSV file containing students.*

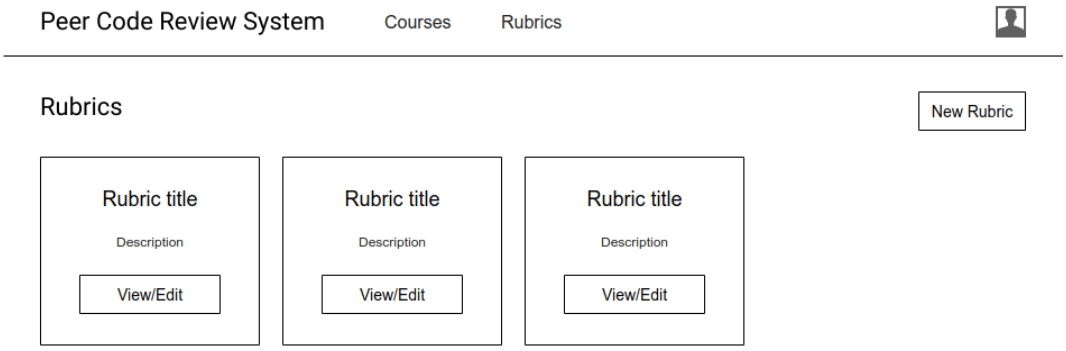


Figure D.8: *The design of the rubrics page, depicted in a wireframe diagram. It shows rubrics an instructor has created. On the top right, it allows an instructor to create a new rubric. Clicking on New Rubric opens a modal similar to the wireframe diagram in Appendix D.2 with fields for rubric title and description.*

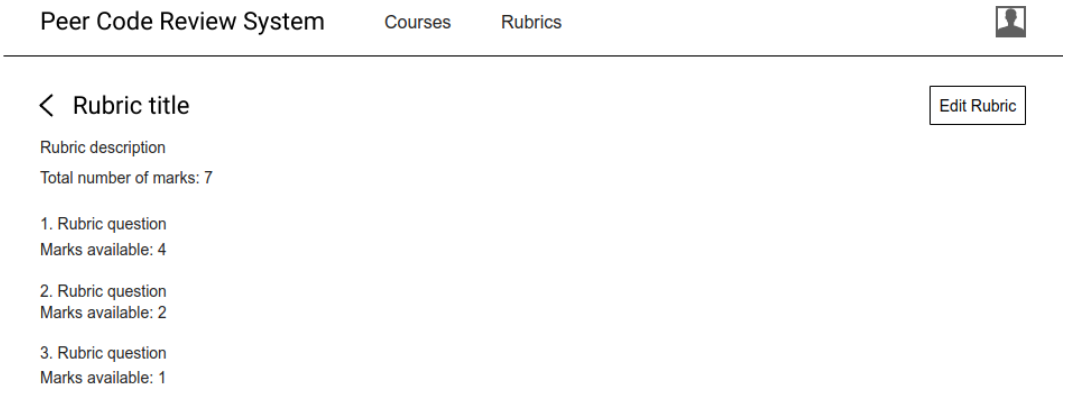


Figure D.9: *The design of the instructor rubric page in preview mode, depicted in a wireframe diagram. It shows a rubric and its questions.*

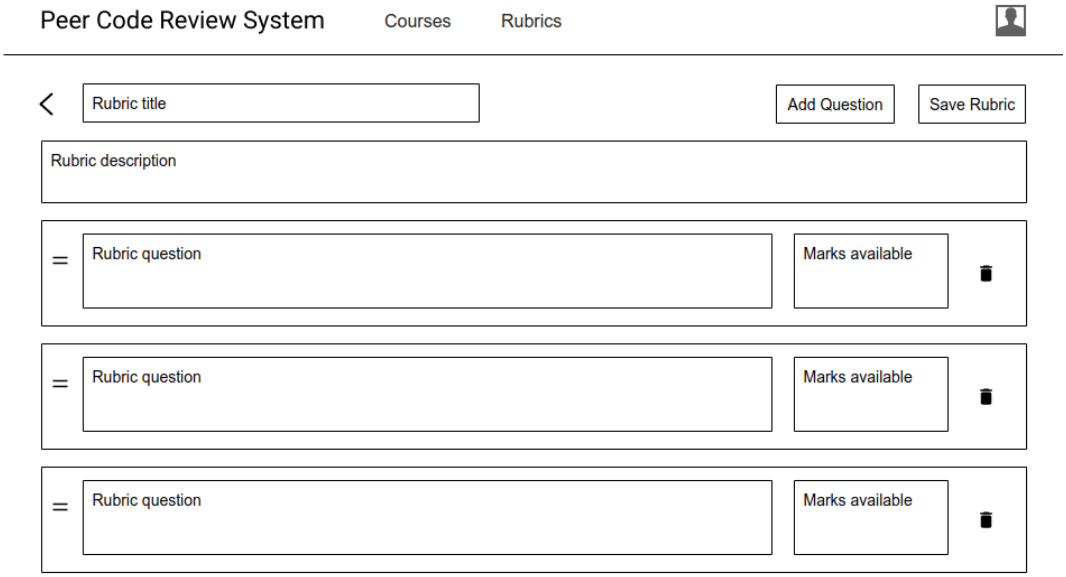


Figure D.10: *The design of the instructor rubric page in edit mode, depicted in a wireframe diagram. It allows an instructor to add, delete, and re-order questions from the rubric. Questions can be re-ordered via drag and drop. It also allows the instructor to edit the rubric title and description.*

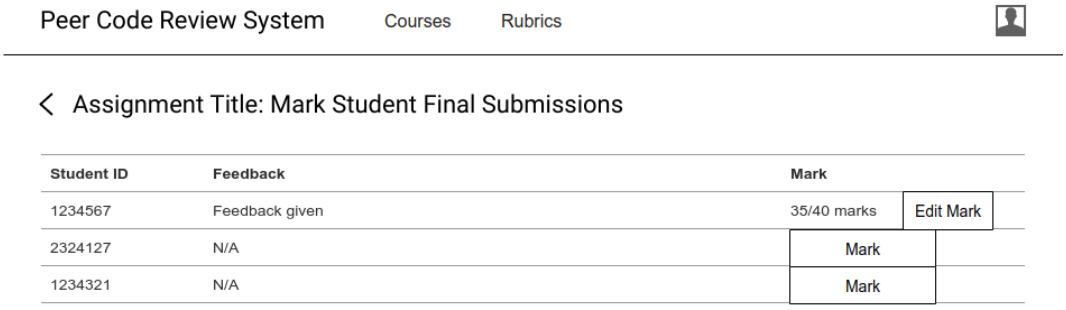


Figure D.11: *The design of the mark student final submissions page, depicted in a wireframe diagram. It shows several students that submitted their final code and allows the course instructor to mark their work.*

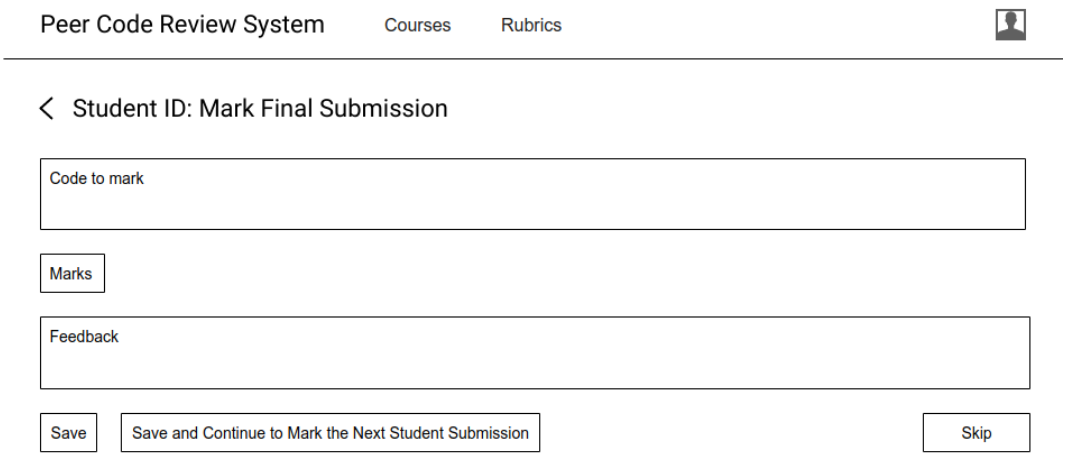


Figure D.12: *The design of the mark final submission page, depicted in a wireframe diagram. It shows a final submission belonging to a student. An instructor can mark the submission and provide feedback. Upon marking, they can save, save and continue to the next student submission, or skip the submission.*

E Student Wireframe Diagrams

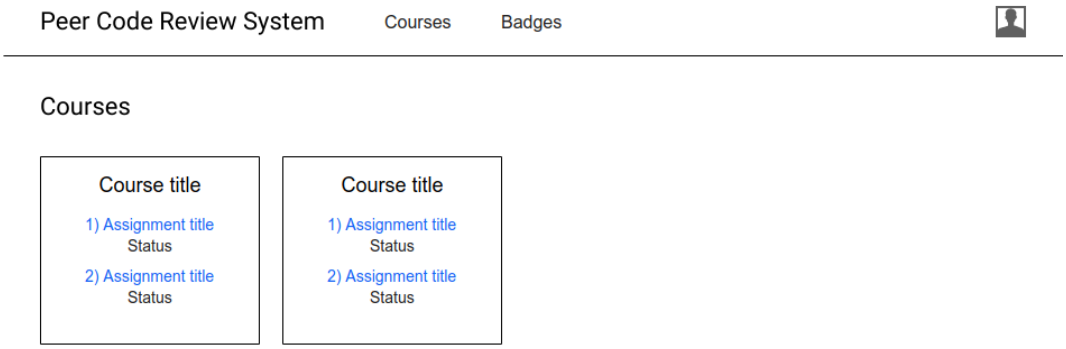


Figure E.1: *The design of the student courses page, depicted in a wireframe diagram. It shows an overview of the courses a student is enrolled in and the assignments within those courses.*

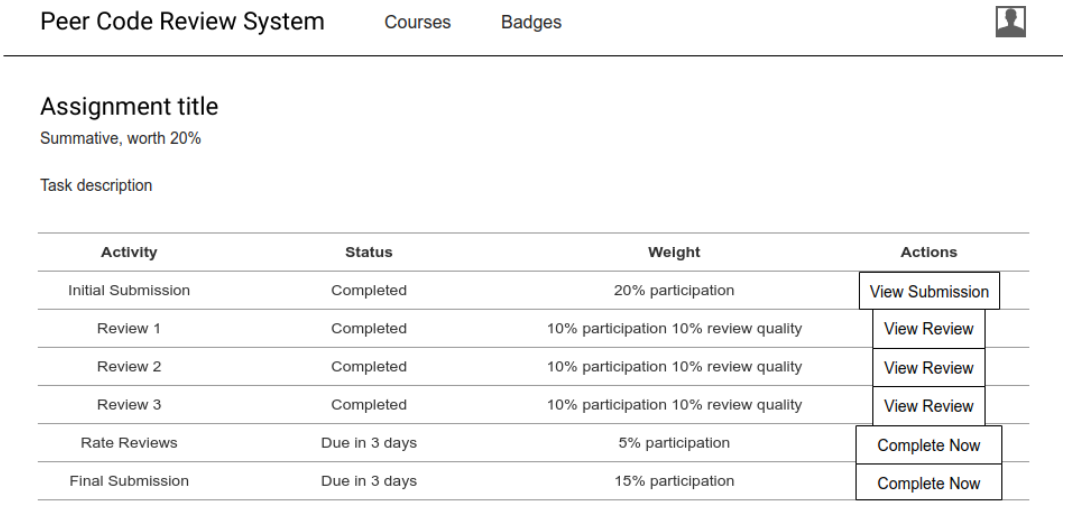


Figure E.2: *The design of the assignment page, depicted in a wireframe diagram. It shows several stages of an assignment and the weight and status of each stage.*

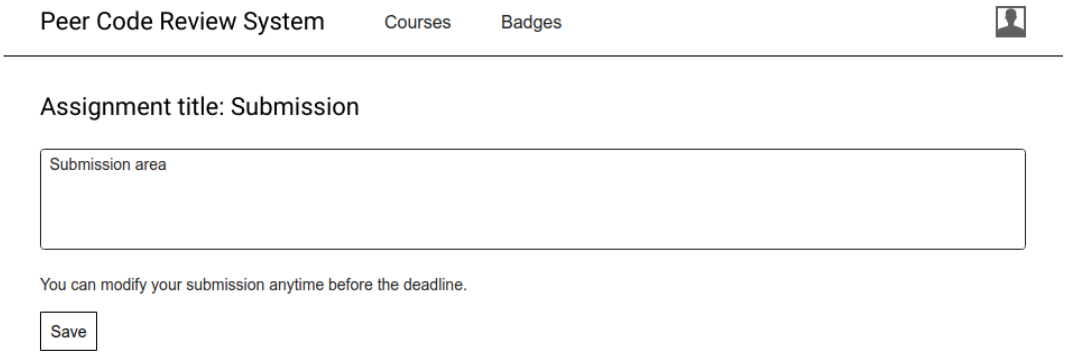


Figure E.3: *The design of the student submit code page, depicted in a wireframe diagram. It provides a submission area for a student to upload their code.*

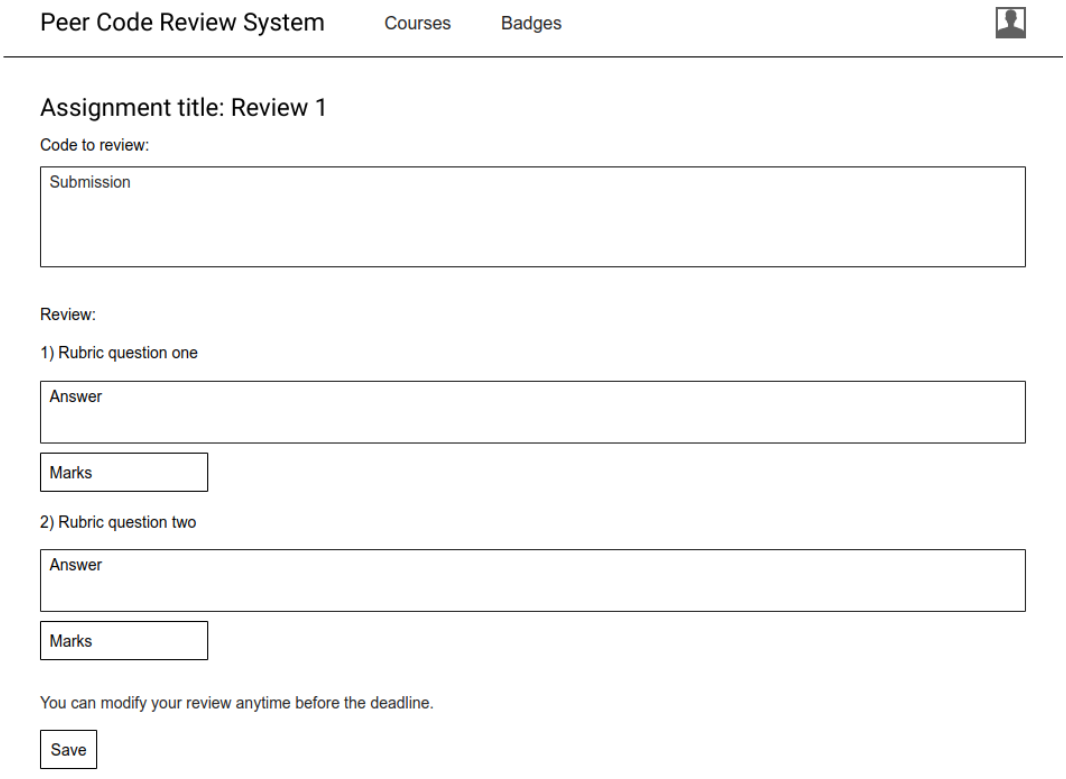


Figure E.4: *The design of the student peer review page, depicted in a wireframe diagram. It shows the code a student has been assigned to review and provides the student with the assignment rubric to complete the review.*

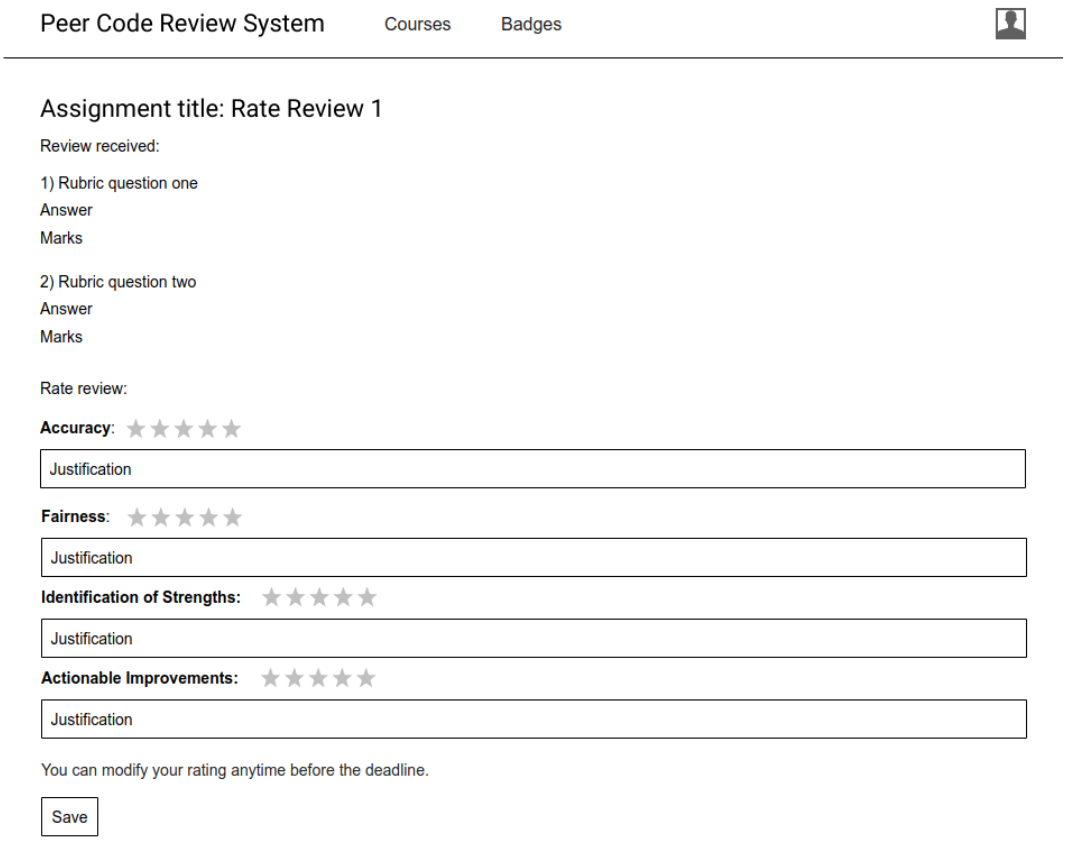


Figure E.5: *The design of the student rate reviews page, depicted in a wireframe diagram. It shows the review a student has received and allows them to rate the review based on four categories. For each category, the student must award the reviewer up to 5 stars and provide a justification for their rating.*

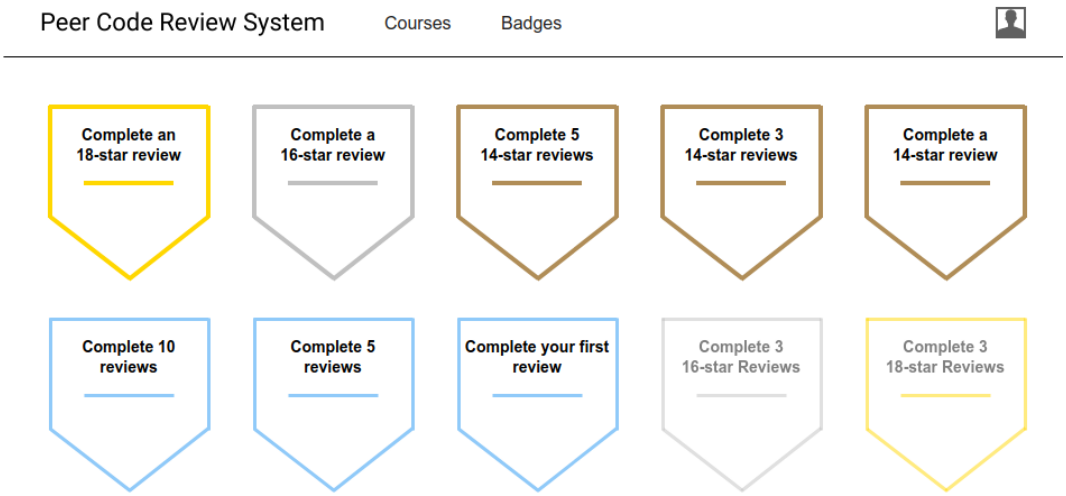


Figure E.6: *The design of the student badges page, depicted as a wireframe diagram. It shows badges a student has been awarded for completing reviews and for high-quality reviews. It also shows two fainter badges. These are badges the student has not been awarded yet.*

F Authentication REST API Endpoints

Authentication endpoints are accessible to both students and instructors. Once a user signs in, the session ID cookie is set and is transmitted with all subsequent requests. All unsuccessful requests have an appropriate error code and message.

F.1 Sign In

Sending an HTTP POST request to **/auth/login/** with user email address and password in the request body. If successful, the response will create a session ID cookie on the client browser, which will be sent with subsequent requests.

F.2 Sign Out

Sending an HTTP POST request to **/auth/logout/** with the session ID in the request body. If successful, the user will be signed out, and the session ID will be removed from cookies and will no longer be valid.

F.3 Change Password

Sending an HTTP POST request to **/auth/password/change/** with the session ID, current password and new password in the request body. If successful, the user's password will be updated.

F.4 Reset Password Request

Sending an HTTP POST request to **/auth/password/reset/** with user email address in the request body. If successful, a password reset email will be sent to the user's email address containing a password reset URL embedded with a unique uid and token.

F.5 Confirm New Password

Sending an HTTP POST request to **/auth/password/reset/confirm/** with a new password, uid, and a token in the request body. If successful, the user's password will be updated.

F.6 Get User Details

Sending an HTTP GET request to **/auth/user/** with the session ID in the request body. If successful, the response will contain the user's email address, given names, surname, and user type (student or instructor).

G Instructor REST API Endpoints

Instructor endpoints are accessible only to instructors. All instructor endpoints require a session ID with the request. If the session ID belongs to a different instructor than the one making the request, an error will be returned. If an instructor attempts to modify content belonging to a different instructor, an error will be returned. Additionally, if attempting to retrieve a resource that does not exist, HTTP 204 is returned.

Note that "instructor_id", "assignment_id", etc., in the request URI indicate that these IDs must be present in the URI of the request.

G.1 Get, Create, Modify, Delete Courses

Sending an HTTP GET request to **/api/courses/**. If successful, the response will contain all courses.

Sending an HTTP GET request to **/api/courses/"course_id"/**. If successful, the response will contain the requested course..

Sending an HTTP POST request to **/api/courses/** with a course in the request body. If successful, the course will be saved.

Sending an HTTP PATCH request to **/api/courses/"course_id"/** with an existing course in the request body. If successful, the course will be updated.

Sending an HTTP DELETE request to **/api/courses/"course_id"/**. If successful, the course will be deleted.

G.2 Manage Students in Course

Request URI: **/api/courses/"course_id"/manage_students/**.

Sending an HTTP GET request. If successful, the response will contain all students in the course.

Sending an HTTP POST request with a list of students in the request body. If successful, the students will be added to the course, and emails will be sent requesting them to set their password (if they do not have an account).

Sending an HTTP POST request with a "delete" flag set to True and a list of student IDs to remove from the course in the request body. If successful, the requested students will be removed from the course.

G.3 Get, Create, Modify, Delete Assignments

Sending an HTTP GET request to **/api/assignments/**. If successful, the response will contain all assignments.

Sending an HTTP GET request to **/api/assignments/"assignment_id"/**. If successful, the response will contain the requested assignment.

Sending an HTTP POST request to **/api/assignments/** with a assignment in the request body. If successful, the assignment will be saved.

Sending an HTTP PATCH request to **/api/assignments/"assignment_id"/** with an existing assignment in the request body. If successful, the assignment will be updated.

Sending an HTTP DELETE request to **/api/assignments/"assignment_id"/**. If successful, the assignment will be deleted.

G.4 Get, Create, Modify, Delete Rubrics

Note, questions within rubrics are handled separately and are referred to as rubric elements.

Sending an HTTP GET request to **/api/rubrics/**. If successful, the response will contain all rubrics.

Sending an HTTP GET request to **/api/rubrics/"rubric_id"/**. If successful, the response will contain the requested rubric.

Sending an HTTP POST request to **/api/rubrics/** with a rubric in the request body. If successful, the rubric will be saved.

Sending an HTTP PATCH request to **/api/rubrics/"rubric_id"/** with an existing rubric in the request body. If successful, the rubric is updated.

Sending an HTTP DELETE request to **/api/rubrics/"rubric_id"/**. If successful, the rubric will be deleted.

G.5 Get, Create, Modify, Delete Questions from Rubrics

Note, rubrics themselves are handled separately.

Request URI: **/api/rubrics/"rubric_id"/elements/**.

Sending an HTTP GET request. If successful, the response will contain the requested rubric with its questions.

Sending an HTTP POST request with a list of questions in the request body. If successful, the questions will be added to the rubric.

Sending an HTTP POST request with a "delete" flag set to True and a list of question IDs to remove from the rubric in the request body. If successful, the requested questions will be removed from the rubric.

G.6 Get All Final Submissions for Assignment

Sending an HTTP GET request to **/api/courses/"course_id"/assignment/"assignment_id"/final_submissions/**. If successful, the response will contain all final submissions for the given assignment. If the final submission stage is ongoing, an error is returned.

G.7 Mark a Final Submission

Sending an HTTP PATCH request to **/api/assignment/"assignment_id"/student/"student_id"/mark_final_submission/** with the marked final submission including number of marks awarded

and feedback fields in the request body. If successful, the marked final submission will be saved.

G.8 Download Assignment Grades

Sending an HTTP GET request to **/api/assignment/"assignment_id" /get_all_assignment_grades/**. If successful, the response will contain grades for every student in the assignment including a full breakdown of the calculation.

H Student REST API Endpoints

Student endpoints are accessible only to students. All student endpoints require a session ID with the request. If the session ID belongs to a different student than the one making the request, an error will be returned. Additionally, if attempting to retrieve a resource that does not exist, HTTP 204 is returned.

Note that "student_id", "assignment_id", etc., in the request URI indicate that these IDs must be present in the URI of the request.

H.1 Accept Invite to Join the System and Set Password

Sending an HTTP POST request to **/set_password/** with a new password, uid, and a token. If successful, the student's password will be set.

H.2 Get Enrolled Courses

Sending an HTTP GET request to **/api/student/"student_id"/courses/**. If successful, the response will contain all courses the student is enrolled in.

H.3 Get Assignments and Get Assignment with ID

Sending an HTTP GET request to **/api/assignments/**. If successful, the response will contain all assignments.

Sending an HTTP GET request to **/api/assignments/"assignment_id"/**. If successful, the response will contain the requested assignment.

H.4 Get Rubric Details

Sending an HTTP GET request to **/api/rubrics/"rubric_id"/elements**. If successful, the response will contain the requested rubric with its questions/elements.

H.5 Get, Create, Modify Assignment Initial Submission

Request URI: **/api/student/"student_id"/assignment/"assignment_id"/initial_submission/**.

Sending an HTTP GET request. If successful, the response will contain the initial submission.

Sending an HTTP POST request containing an initial submission in the request body. If successful, the initial submission has been created. If the assignment is not accepting submissions, an error is returned.

Sending an HTTP PATCH request containing an initial submission with an initial submission ID in the request body. If successful, the initial submission has been updated. If the assignment is not accepting submissions, an error is returned.

H.6 Get Reviews

Sending an HTTP GET request to **/api/student/"student_id"/assignment/"assignment_id" /review/**. If successful, the response will contain three reviews the student must complete.

H.7 Complete a Review

Sending an HTTP PATCH request to **/api/student/"student_id"/assignment/"assignment_id" /review/"review_id"/** with the review in the request body. If successful, the review will be updated. If the assignment is not accepting reviews, an error is returned.

H.8 Get Received Reviews

Sending an HTTP GET request to **/api/student/"student_id"/assignment/"assignment_id" /completed_review/**. If successful, the response will contain three reviews the student has received from other students.

H.9 Rate a Received Review

Sending an HTTP PATCH request to **/api/student/"student_id"/assignment/"assignment_id" /completed_review/"review_id"/** with the review in the request body. If successful, the review will be updated with the given ratings and justifications. If the assignment is not accepting review ratings, an error is returned.

H.10 Get, Create, Modify Assignment Final Submission

Request URI: **/api/student/"student_id"/assignment/"assignment_id"/final_submission/**.

Sending an HTTP GET request. If successful, the response will contain the final submission.

Sending an HTTP POST request containing a final submission in the request body. If successful, the final submission has been created. If the assignment is not accepting submissions, an error is returned.

Sending an HTTP PATCH request containing a final submission with a final submission ID in the request body. If successful, the initial final has been updated. If the assignment is not accepting submissions, an error is returned.

H.11 Get Results for Assignment

Sending an HTTP GET request to **/api/student/"student_id"/assignment/"assignment_id" /get_grades/**. If successful, the response will contain the grades the student has received for the assignment.

H.12 Get All Reviews from Completed Assignments

Sending an HTTP GET request to **/api/student/"student_id"/all_reviews/**. If successful, the response will contain all reviews the student has completed for all completed assignments.

I Implementation

I.1 Instructor Create Assignment

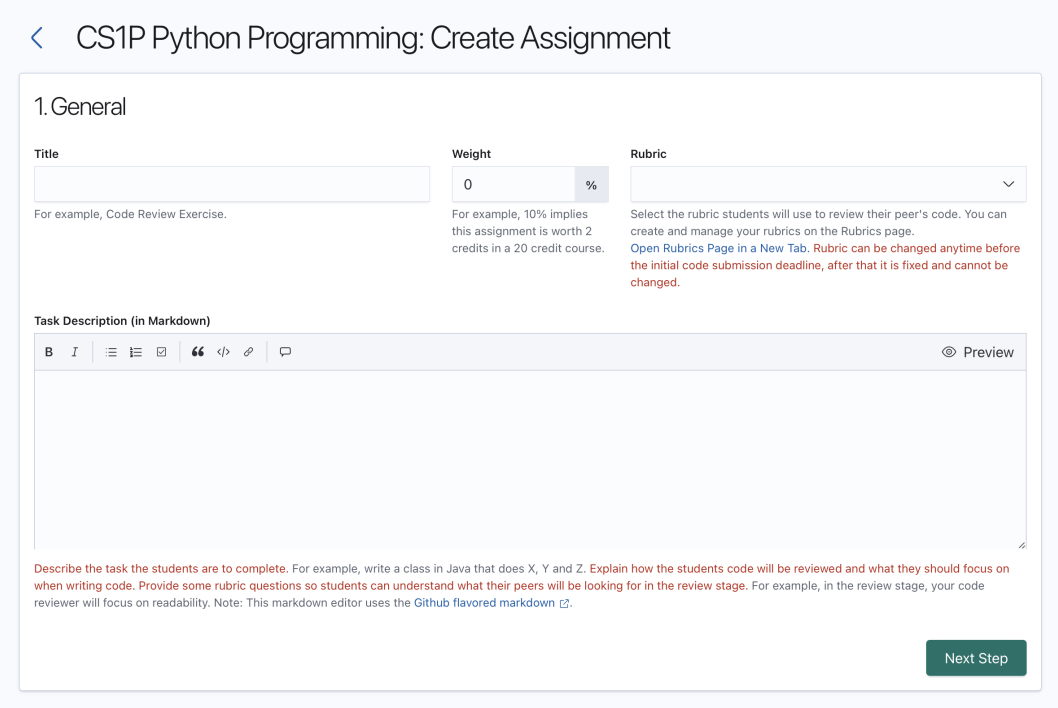


Figure I.1: *The create new assignment page on step 1 of 5. It shows several fields including the assignment description which allows the instructor to format the description using Markdown.*

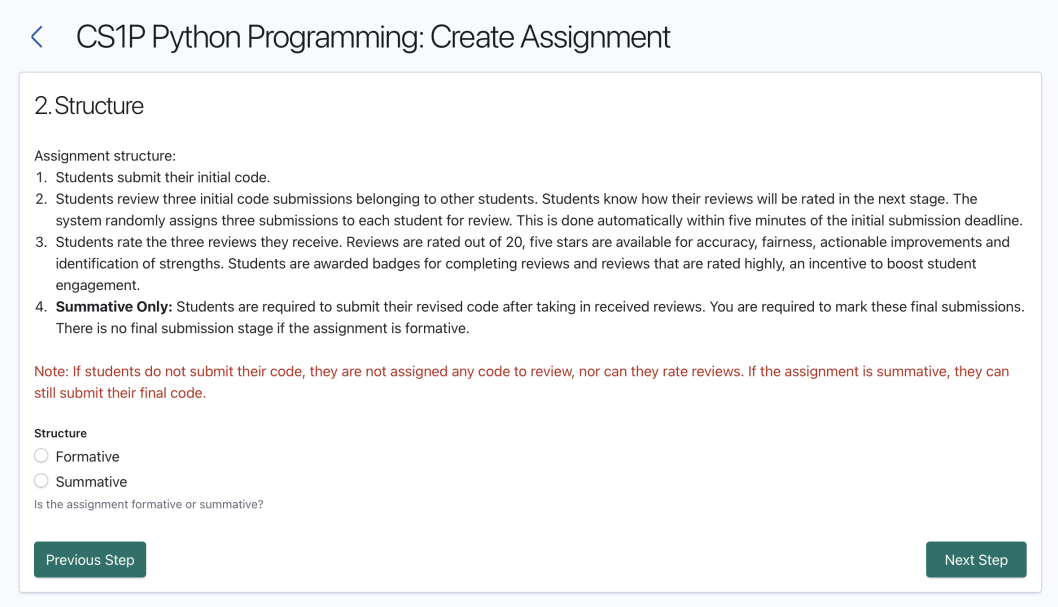


Figure I.2: *The create new assignment page on step 2 of 5. It outlines the structure of formative and summative assignments and allows an instructor to choose the assignment structure.*

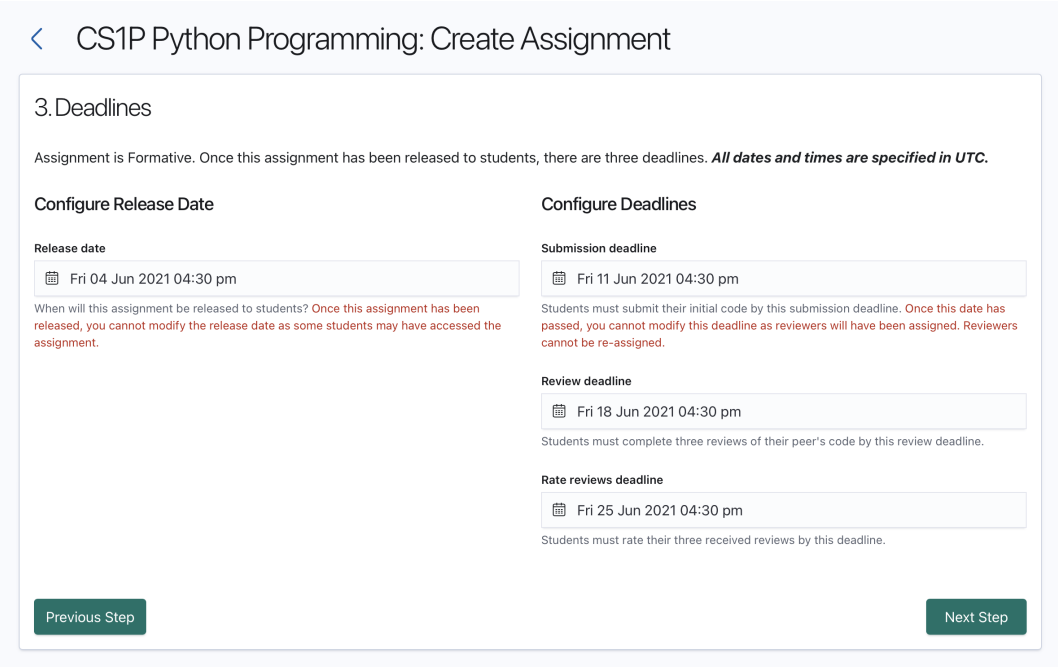


Figure I.3: *The create new assignment page on step 3 of 5. It shows several date and time fields and allows an instructor to set the assignment release date and configure deadlines for each stage of the assignment.*

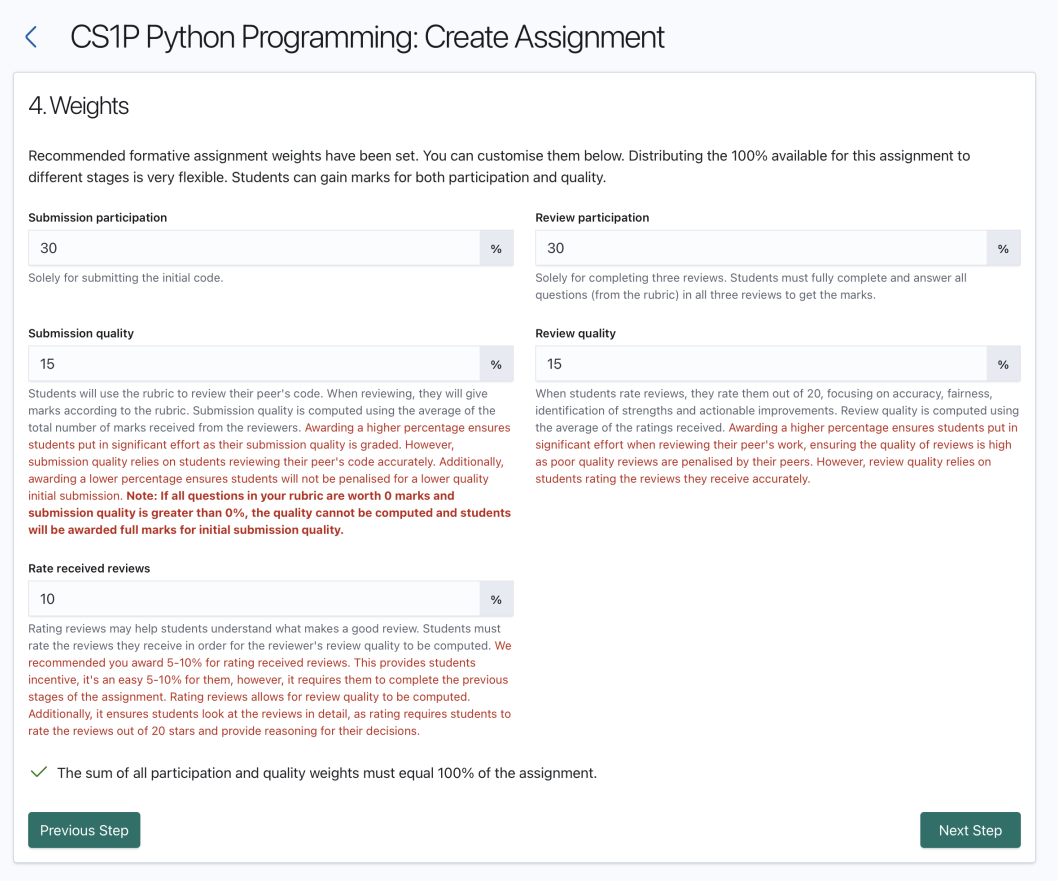


Figure I.4: *The create new assignment page on step 4 of 5. It shows several weight fields which allow an instructor to configure how students are graded. For example, it shows students will receive 30% of the marks available for submitting their code. This step provides great flexibility for assignment weight configuration.*

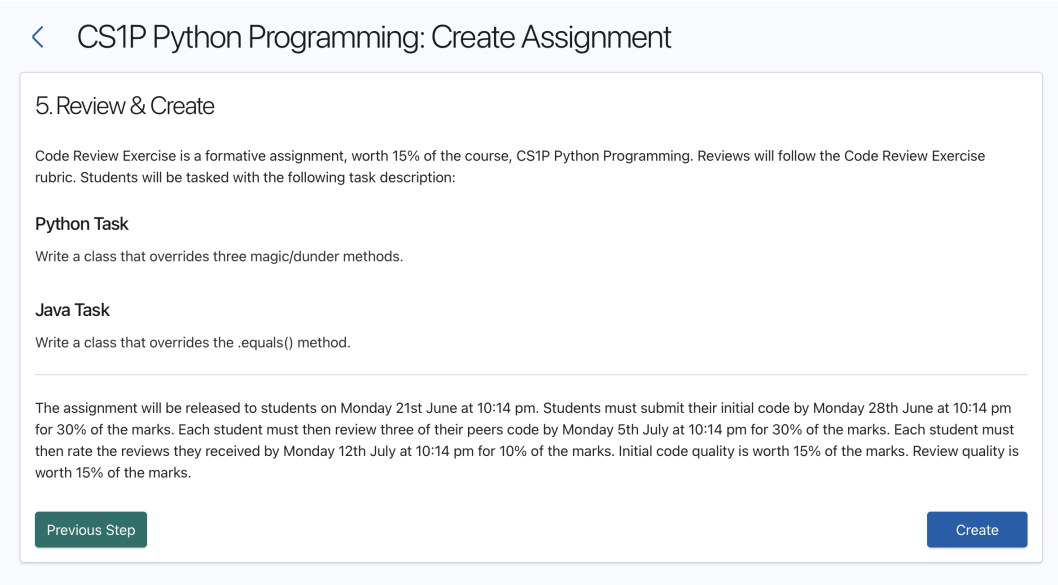


Figure I.5: *The create new assignment page on step 5 of 5. It shows a summary of all data an instructor has already input, in a read-only format. Its sole purpose is to help instructors avoid mistakes.*

I.2 Student Peer Code Review Exercise

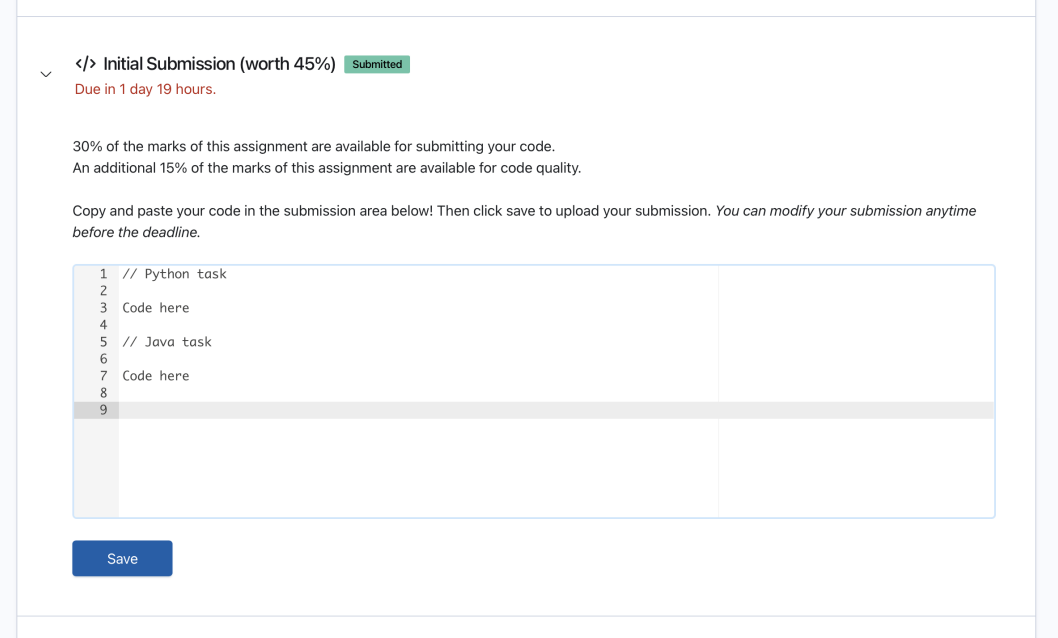


Figure I.6: *The student assignment page showing the initial submission stage. It shows a submission area, which allows a student to upload and submit their code, and modify their submission anytime before the deadline.*

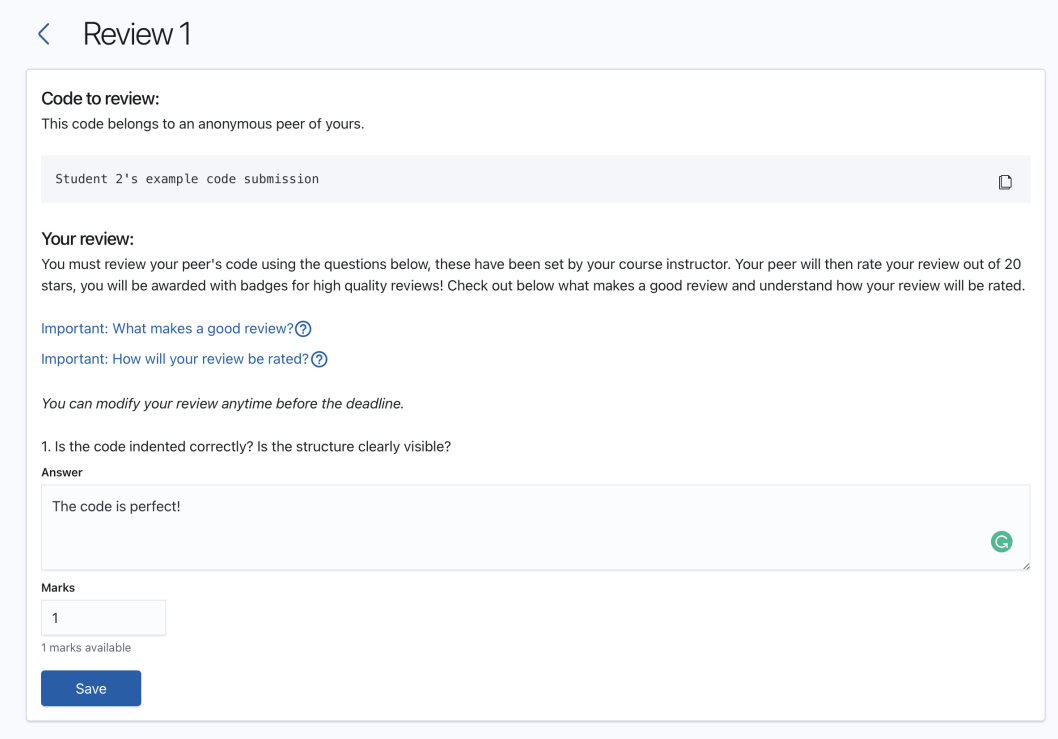


Figure I.7: *The review page showing the first review a student must complete. It shows the code a student must review and allows the student to submit their review following the assignment rubric.*

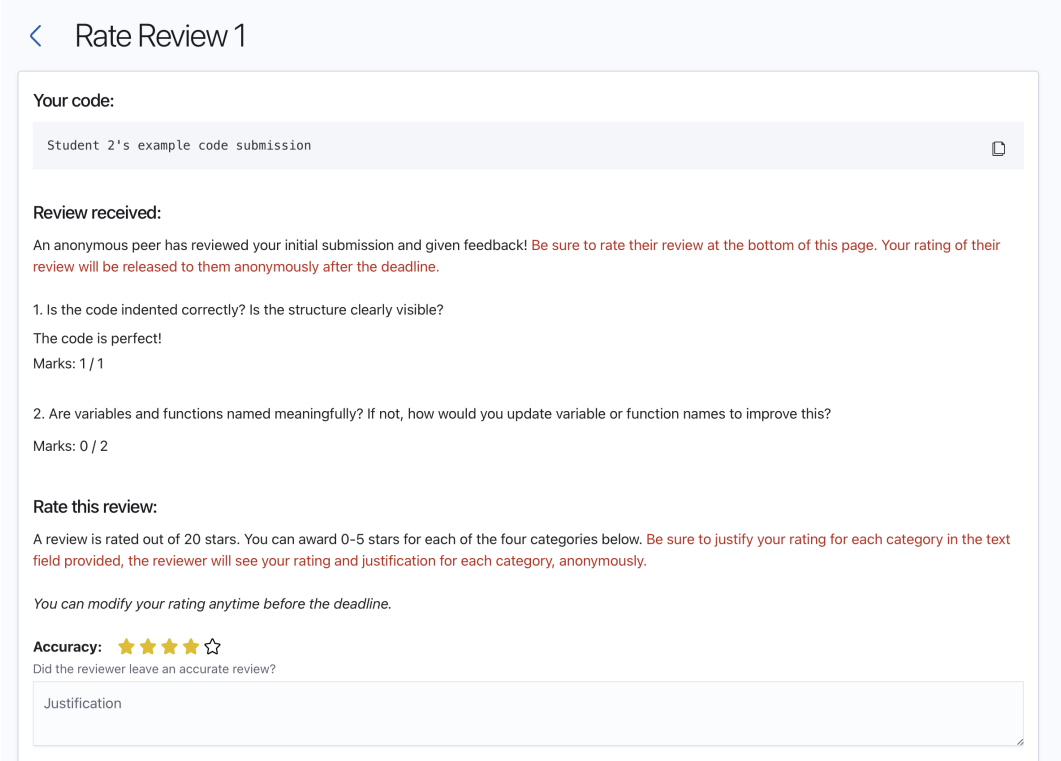


Figure I.8: *The first half of the rate review page showing a student's code and the review they have received. It shows a rate review section where the student can rate the reviewer based on accuracy. Figure I.9 shows the second half of the page.*

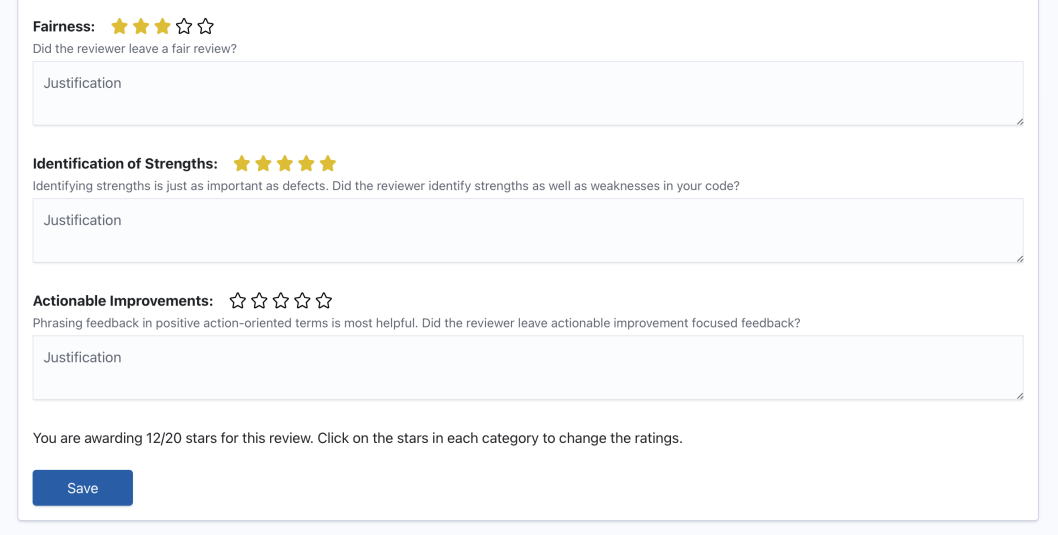


Figure I.9: *The second half of the rate review page, that allows a student to rate a review based on accuracy, fairness, identification of strengths, and actionable improvements. Figure I.8 shows the first half of the page.*

I.3 Student Badges

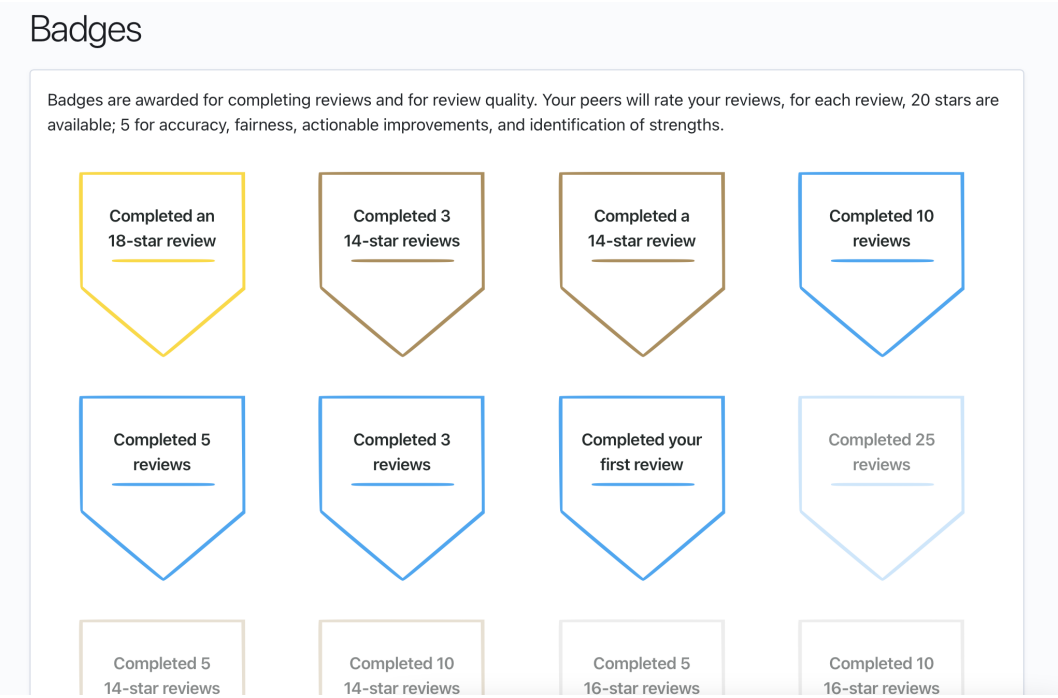


Figure I.10: *The student badges page showing several badges a student has been awarded, for completing reviews and high-quality reviews. It shows gold and bronze badges awarded for 18-star and 14-star reviews. Fainter badges are badges that have not been awarded yet.*

J | Deployed Application Access Details

Below is one instructor and four student accounts. The URL to try out the application: https://peer-code-review-system.herokuapp.com. Use these student details if you wish to add students to courses. Please do not change account passwords or add new students to this deployment.

Instructor account:

Email address: instructor@peer-code-review-system.com

Password: Ps1WxLpa

Student ID	Given names	Surname	Email address	Password
1111111	John	Doe	student1@peer-code-review-system.com	1pWwxSPA
2222222	Jane	Doe	student2@peer-code-review-system.com	10TlsSPA
3333333	Mary	Smith	student3@peer-code-review-system.com	MQAxU9ij
4444444	George	Brown	student4@peer-code-review-system.com	9bMki83e

Table J.1: *Student accounts on the deployment of the peer code review system.*

K Acceptance Tests

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
SIGN IN	A student or instructor enters their email address and password and clicks on Sign In.	The student or instructor is successfully authenticated and redirected to the instructor or student courses page.	As expected, passed.
SIGN OUT	A user clicks on Sign Out.	The user is no longer authenticated and is redirected to the sign in page.	As expected, passed.
FORGOT PASSWORD 1	A user enters a valid email address and clicks Request Password Reset Email.	An email is sent to the user containing a password reset link. If the email address is unknown to the system, no email is sent.	As expected, passed.
FORGOT PASSWORD 2	A user opens the password reset link from their forgot password email, enters a new password, and enters the new password again and clicks Confirm.	The student's account password is reset, a success message is displayed, and the user is redirected to sign in page.	As expected, passed.
CHANGE PASSWORD	A user enters their old password, a new password, and the new password again, and clicks Save.	The user's account password is updated, a success message is displayed. The user remains signed in.	As expected, passed.
ADMINISTRATOR: CREATE NEW INSTRUCTOR ACCOUNT	An administrator navigates to the admin site at /admin/, and clicks on Instructors, then on Add Instructor, enters an instructor id and creates a user account by setting their name, email address, password, and checking the box "is instructor".	The instructor is added to the system. They can now sign into the system. They can now change their password as it was set by the administrator.	As expected, passed.
INSTRUCTOR: CREATE COURSE	Click New Course, enter a title and number of credits and click Save.	A course is created and added to the instructor's list of courses on the courses page.	As expected, passed.
INSTRUCTOR: VIEW COURSES	Navigate to courses page.	A list of courses the instructor has created is shown.	As expected, passed.
INSTRUCTOR: ADD A STUDENT TO COURSE	Click Manage Students in a course, click Add Student, enter student id, given names, surname, and email address, and click Save.	A student is added to the course. If they are unknown to the system, a welcome email requesting them to set their password is sent.	As expected, passed.

Figure K.1: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
INSTRUCTOR: ADD STUDENTS TO COURSE VIA CSV	Click Manage Students in a course, click Add Students via CSV, and upload a CSV file containing student details. Student id in column 1, and student surname and given names in column 2, separated by a comma. E.g. 1234567,"Doe,John Michael". Enter a postfix for the system to generate email addresses, e.g. @student.gla.ac.uk	All students are added to the course. If a student is unknown to the system, a welcome email requesting them to set their password is sent to them.	As expected, passed.
INSTRUCTOR: REMOVE STUDENTS FROM COURSE	Click Manage Students in a course, select the students to remove by clicking the checkboxes next their name/student id. Click Delete Selected Students.	Selected students are removed from the course immediately.	As expected, passed.
INSTRUCTOR: CREATE FORMATIVE AND SUMMATIVE ASSIGNMENT	Click New Assignment in a course, enter assignment details through five assignment creation steps, click Create.	The assignment is created.	As expected, passed.
INSTRUCTOR: SET ASSIGNMENT WEIGHTS WHEN CREATING ASSIGNMENT	In step 4 of create assignment page, specify the participation and quality weights for each stage of the assignment.	The assignment has weights for grading.	As expected, passed.
INSTRUCTOR: SET ASSIGNMENT DEADLINES WHEN CREATING ASSIGNMENT	In step 3 of create assignment page, specify the release date of the assignment and deadlines for each stage.	The assignment has a release date and deadlines set.	As expected, passed.
INSTRUCTOR: VIEW ASSIGNMENTS IN EACH COURSE	Navigate to courses page.	Beneath each course, the course assignments are listed.	As expected, passed.
INSTRUCTOR: CREATE RUBRIC	Click on New Rubric and enter a rubric title and description, then click Save.	A new rubric is created.	As expected, passed.

Figure K.2: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
INSTRUCTOR: VIEW RUBRICS	Navigate to the rubrics page.	The rubrics are listed. The instructor can click on a rubric to open and view details.	As expected, passed.
INSTRUCTOR: ADD QUESTIONS TO THE RUBRIC	Open a rubric, click Edit Rubric. Click Add Question to add a question to the rubric, providing the question text and number of marks available, then click Add to Rubric. Then click save changes.	A question is added to the rubric.	As expected, passed.
INSTRUCTOR: REMOVE QUESTIONS FROM THE RUBRIC	Open a rubric, click Edit Rubric. Click the trash can icon next to a question. Then click save changes.	The question is removed from the rubric.	As expected, passed.
INSTRUCTOR: MARK AND GIVE FEEDBACK TO A STUDENT'S FINAL SUBMISSION	Click Mark Final Submissions. Select a student whose code to mark. View their code, enter feedback and number of marks awarded, then click Save.	The number of marks awarded and feedback for the student's final submission is saved.	As expected, passed.
INSTRUCTOR: MODIFY MARKS AND FEEDBACK GIVEN TO A STUDENT'S FINAL SUBMISSION	Click Mark Final Submissions. Choose a student whose mark needs editing, click Edit mark. Update the feedback and number of marks awarded, then click Save.	The number of marks awarded and feedback for the student's final submission is updated.	As expected, passed.
INSTRUCTOR: DOWNLOAD STUDENT FINAL MARKS WITH A BREAKDOWN OF RESULTS	Click Download Results for a completed assignment.	A CSV file is downloaded containing a breakdown of results for every student in the course for that assignment.	As expected, passed.
INSTRUCTOR: EDIT COURSE DETAILS	Click the pencil icon next to the course title. Edit the course title and number of credits in the modal. Click save.	The course title and number of credits are updated.	As expected, passed.
INSTRUCTOR: DELETE COURSE	Click the trash-can icon next to the course title. If the course has no students and no assignments. Click confirm to dismiss the warning message and delete the assignment.	The course is deleted.	As expected, passed.

Figure K.3: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
INSTRUCTOR: EDIT ASSIGNMENT BEFORE RELEASE DATE	Click the pencil icon next to the assignment. Edit any of the assignment details using the five steps and click Save to update the assignment.	The assignment is updated.	As expected, passed.
INSTRUCTOR: EDIT ONGOING ASSIGNMENT	Click the pencil icon next to the assignment. You cannot edit the release date after the assignment is released. You cannot edit the submission deadline after reviewers have been assigned.	The assignment release date field is disabled after the release date. The submission deadline field is disabled once reviewers have been assigned.	As expected, passed.
INSTRUCTOR DELETE ASSIGNMENT	Click the delete icon next to the assignment. Click confirm to dismiss the warning and delete the assignment.	The assignment is deleted.	As expected, passed.
INSTRUCTOR: MODIFY RUBRIC	Select the rubric to modify and click Edit Rubric. Make any changes and click Save Changes.	The rubric is updated.	As expected, passed.
INSTRUCTOR: DELETE RUBRIC	Select the rubric to delete and click Edit Rubric. Click Delete Rubric and confirm.	The rubric is deleted.	As expected, passed.
INSTRUCTOR: RE-ORDER QUESTIONS WITHIN A RUBRIC	Select the rubric and click Edit Rubric. Drag and drop questions within the rubric and click Save Changes.	The rubric questions are re-ordered.	As expected, passed.
INSTRUCTOR: SET WEIGHTINGS FOR QUESTIONS IN A RUBRIC	Select the rubric and click Edit Rubric. Enter marks next to each question and click Save Changes.	The rubric questions weightings are set.	As expected, passed.
INSTRUCTOR: PUBLISH RESULTS WITHIN THE SYSTEM	Once an assignment is completed, click Publish Results and click Confirm.	The results are published and accessible to students.	As expected, passed.
INSTRUCTOR: VIEW WHICH STUDENTS HAVE JOINED THE SYSTEM FOR A GIVEN COURSE	Click Manage Students in a course to view a list of students in the course.	Students with status Pending have not joined the system. Students with status Ready have joined the system.	As expected, passed.

Figure K.4: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
INSTRUCTOR: FORMAT TASK DESCRIPTION USING MARKDOWN	When creating or editing an assignment, the task description can be specified in Markdown.	A markdown editor allows the task description to be formatted appropriately.	As expected, passed.
STUDENT: JOIN THE SYSTEM VIA EMAIL INVITE.	Click on the link in the email received and set account password by typing in a password and confirm the password, then click Confirm.	Student account has been set up, they can now sign in.	As expected, passed.
STUDENT: VIEW COURSES	Navigate to the courses page.	A list of courses is shown.	As expected, passed.
STUDENT: VIEW ASSIGNMENTS.	Navigate to the courses page.	Beneath each course, a list of assignments is shown.	As expected, passed.
STUDENT: VIEW ASSIGNMENT DETAILS	Click on an assignment on the courses page.	Opens the assignment and shows all assignment details.	As expected, passed.
STUDENT: SUBMIT CODE BEFORE THE INITIAL SUBMISSION DEADLINE.	Open the assignment, click on Initial Submission to expand the section, enter code into the submission area and click Save.	The student's initial submission is uploaded/saved.	As expected, passed.
STUDENT: REVIEW ASSIGNED SUBMISSIONS USING RUBRIC BEFORE REVIEW DEADLINE.	Open the assignment, click on Complete Peer Reviews to expand the section, select one of three reviews to complete. Follow the rubric to review peer code, then click Save.	The peer review is saved.	As expected, passed.
STUDENT: VIEW REVIEWS RECEIVED FROM PEERS AFTER REVIEW DEADLINE.	Open the assignment, click on View and Rate Received Reviews to expand the section, select one of three reviews to view. View the received review.	The student has viewed the received review.	As expected, passed.

Figure K.5: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
STUDENT: RATE RECEIVED REVIEWS AND GIVE FEEDBACK TO PEERS BEFORE RATE REVIEW DEADLINE.	Open the assignment, click on View and Rate Received Reviews to expand the section, select one of three reviews to rate. Rate the received review based on accuracy, fairness, identification of strengths, and actionable improvements, and justify ratings. Then click Save.	The student has rated the received review.	As expected, passed.
STUDENT: SUBMIT FINAL REVISED CODE BEFORE THE FINAL SUBMISSION DEADLINE	Open the assignment, click on Final Submission to expand the section, enter code into the submission area and click Save.	The student's final submission is uploaded/saved.	As expected, passed.
STUDENT: VIEW BADGES AWARDED	Navigate to the badges page.	The student can see all the badges they have been awarded for completing reviews and for high-quality reviews.	As expected, passed.
STUDENT: GIVE MARKS TO A PEER'S SUBMISSION WHEN REVIEWING THEIR CODE.	Open the assignment, click on Complete Peer Reviews to expand the section, select one of three reviews to complete. Follow the rubric to review and mark peer code, then click Save.	The peer review/assessment is saved.	As expected, passed.
STUDENT: MODIFY SUBMISSIONS, REVIEWS, AND RATINGS	Open the assignment, click on any stage and modify the submission any time before that deadline.	The submission, review, and rating is updated.	As expected, passed.
STUDENT: VIEW RESULTS ON THE SYSTEM.	Open the assignment once results have been published by the instructor.	The student has viewed a breakdown of their results and the overall final mark for the assignment.	As expected, passed.

Figure K.6: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

TEST	DESCRIPTION/ACTION	EXPECTED RESULT	ACTUAL RESULT
SYSTEM: INVITE STUDENTS TO JOIN SYSTEM VIA EMAIL IF STUDENT IS UNKNOWN TO THE SYSTEM.	When students are added to the a course, if the student is unknown to the system, the system emails them requesting them to set their account password. Adding several known and unknown students to a course.	The system sends emails to students not registered with the system. The system does not email students who already have accounts set up.	As expected, passed.
SYSTEM: ENSURE ANONYMITY BETWEEN STUDENTS	Ensure student ids and personally identifiable information such as the student's name is not revealed to peers.	Peers must review code from an "anonymous peer". There are no references to the student that can reveal their identity.	As expected, passed.
SYSTEM: AUTOMATICALLY AND RANDOMLY ALLOCATE REVIEWERS ON ASSIGNMENT SUBMISSION DEADLINE.	After the initial code submission deadline, the system automatically and randomly allocates students to review code.	Within two minutes after the submission deadline, the student has code to review.	As expected, passed.
SYSTEM: TIME ZONE SUPPORT	Peers can partake in the assignment from different time zones. Update the time zone on the device and check to ensure deadlines are correct.	All students have the same deadline regardless of geographical location.	As expected, passed.
SYSTEM: DARK MODE TOGGLE	Click on the dark/light mode toggle.	The system switches between light and dark mode.	As expected, passed.

Figure K.7: *The result of several acceptance tests conducted on the finished peer code review system. Each test defines the expected behaviour of a feature and the actual result.*

L Pilot User Evaluation Tasks

Instructor tasks:

1. Sign in for the first time using the credentials provided and change your password.
2. The system has a light mode and a dark mode. Try switching between them.
3. Create a rubric containing two questions.
4. Create a course.
5. Create a dummy formative assignment for your course. Then try to edit and change the assignment title.
6. Add a student to your course using the following credentials, student ID 2324127, given names Mohammad, surname Majid – this is because the system will send an email to the student.
7. An existing course, Functional Programming, has one assignment, Code Review Exercise. It is a formative assignment and has been completed by two students (test accounts). Try downloading their grades and publishing the assignment results.
8. Sign out from the system.

Student tasks:

1. You have received a Welcome email asking you to join the peer code review system. Follow instructions in the email, then sign into the system.
2. The system has a light mode and a dark mode. Try switching between them.
3. Browse through your courses and assignments.
4. Complete the assignment titled "Pilot Evaluation". This involves submitting initial code, completing a peer review, and rating any reviews you received.
5. Check your results for the "Pilot Evaluation" assignment you just completed.
6. Explore the badges page.
7. Sign out from the system.

M | Final User Evaluation Assignment

Task Description:

In Python, write the following two functions:

1. A function that takes a name a prints "Hello {name}".
2. A function that takes a list of integers and prints the first integer in the list.

Figure M.1: *The task description of the assignment used in the final user evaluation for students.*

1. Is the code indented correctly? Is the structure clearly visible?
Marks available: 1

2. Are variables and functions named meaningfully? If not, how would you update variable or function names to improve this?
Marks available: 2

Figure M.2: *The assignment rubric used in the final user evaluation for students.*

N System Usability Questions

The System Usability Scale questions, developed by Brooke (1995), consist of the following ten statements:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

O System Usability Student Results

Participant	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Total Points	SUS Score
Student 1	5	5	6	6	5	4	5	6	3	45	83.33
Student 2	5	6	6	5	6	6	6	6	5	51	94.44
Student 3	5	5	6	6	6	6	4	5	4	47	87.04
Student 4	5	6	5	2	5	5	6	6	6	46	85.19
Student 5	6	6	6	6	6	6	6	6	6	54	100
Student 6	6	6	6	6	6	6	6	6	6	54	100
Student 7	6	6	6	6	6	6	6	6	6	54	100
Student 8	5	5	6	6	5	6	5	6	5	49	90.74
Student 9	5	6	5	5	6	6	6	4	5	48	88.89
Student 10	6	6	6	6	6	6	6	6	6	54	100
Student 11	5	6	6	6	6	6	6	6	5	52	96.30
Student 12	6	6	6	5	5	6	5	5	6	50	92.60
Student 13	5	5	6	6	6	6	6	5	6	51	94.45
Student 14	4	4	6	5	6	5	5	4	5	44	81.48
Student 15	6	6	6	5	6	6	6	5	5	51	94.45
Student 16	5	5	4	5	6	5	5	4	4	43	79.63
Student 17	6	6	5	6	6	6	6	6	6	53	98.15

Table O.1: *The points calculated from the results of the System Usability Scale questionnaire for students. It shows the total points and SUS score for each student. The average SUS score is 92.*

7 Bibliography

Aropa, 2021. URL http://www.dcs.gla.ac.uk/~hcp/aropa/. Online; accessed 10 June 2021.

Atlassian. Git feature branch workflow: Atlassian git tutorial, 2021. URL https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow. Online; accessed 20 June 2021.

K. Beck. *Test-driven development by example*. Addison-Wesley, 2015.

K. Betteridge. What every client should know about sus scores, 2021. URL https://www.bentley.edu/centers/user-experience-center/what-every-client-should-know-about-sus-scores. Online; accessed 18 June 2021.

J. Bowes. Agile vs waterfall - comparing project management methods, Jul 2014. URL https://manifesto.co.uk/agile-vs-waterfall-comparing-project-management-methodologies/. Online; accessed 20 June 2021.

J. Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

Celery. First steps with django for celery integration, 2021. URL https://docs.celeryproject.org/en/stable/django/first-steps-with-django.html. Online; accessed 12 June 2021.

K. Cho, C. Schunn, and R. Wilson. Validity and reliability of scaffolded peer assessment of writing from instructor and student perspectives. *Journal of Educational Psychology*, 98:891–901, 2006.

Django. Django web framework, 2021. URL https://www.djangoproject.com/. Online; accessed 15 June 2021.

Elastic. Elastic ui framework, 2021. URL https://elastic.github.io/eui/#/. Online; accessed 15 June 2021.

M. Fowler. Continuous integration, 2006. URL https://www.martinfowler.com/articles/continuousIntegration.html. Online; accessed 15 June 2021.

S. Gielen, L. Tops, F. Dochy, P. Onghena, and S. Smeets. A comparative study of peer and teacher feedback and of various peer feedback forms in a secondary school writing curriculum. *British Educational Research Journal*, 36(1):143–162, 2010. doi: 10.1080/01411920902894070. URL https://doi.org/10.1080/01411920902894070.

GitHub, 2021. URL https://github.com/. Online; accessed 10 June 2021.

C. D. Hundhausen, A. Agrawal, and P. Agarwal. Talking about code: Integrating pedagogical code reviews into early computing courses. *ACM Trans. Comput. Educ.*, 13(3), Aug. 2013. doi: 10.1145/2499947.2499951. URL https://doi.org/10.1145/2499947.2499951.

T. D. Indriasari, A. Luxton-Reilly, and P. Denny. A review of peer code review in higher education. *ACM Trans. Comput. Educ.*, 20(3), Sept. 2020. doi: 10.1145/3403935. URL https://doi.org/10.1145/3403935.

V. Joshi. Council post: Seven reasons why a website's front-end and back-end should be kept separate, Jul 2018. URL https://www.forbes.com/sites/forbestechcouncil/2018/07/19/seven-reasons-why-a-websites-front-end-and-back-end-should-be-kept-separate/. Online; accessed 16 June 2021.

R. Lu and L. Bol. A comparison of anonymous versus identifiable e-peer review on college student writing performance and the extent of critical feedback. *Journal of Interactive Online Learning*, 6:100–115, 06 2007.

K. Lundstrom and W. Baker. To give is better than to receive: The benefits of peer review to the reviewer's own writing. *Journal of Second Language Writing*, 18(1):30–43, 2009. ISSN 1060-3743. doi: https://doi.org/10.1016/j.jslw.2008.06.002. URL https://www.sciencedirect.com/science/article/pii/S1060374308000313.

R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, USA, 1 edition, 2008. ISBN 0132350882.

J. McGonigal. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. Penguin Group , The, 2011. ISBN 1594202850.

G. Morales-Martinez, P. Latreille, and P. Denny. Nationality and gender biases in multicultural online learning environments: The effects of anonymity. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376283. URL https://doi.org/10.1145/3313831.3376283.

R. Mulder, J. Pearce, and C. Baik. Peer review in higher education: Student perceptions before and after participation. *Active Learning in Higher Education*, 15:157–171, 2014.

S. Nicholson. A recipe for meaningful gamification. 2015.

E. Panadero, M. Romero, and J.-W. Strijbos. The impact of a rubric and friendship on peer assessment: Effects on construct validity, performance, and perceptions of fairness and comfort. *Studies in Educational Evaluation*, 39(4):195–203, 2013. ISSN 0191-491X. doi: https://doi.org/10.1016/j.stueduc.2013.10.005. URL https://www.sciencedirect.com/science/article/pii/S0191491X13000497.

J. Pearce, R. Mulder, and C. Baik. *Involving students in peer review: case studies and practical strategies for university teaching*. Centre for the Study of Higher Education, University of Melbourne, 2009.

Peergrade. Peergrade - engaging student peer review, 2021. URL https://www.peergrade.io/. Online; accessed 13 June 2021.

H. Purchase and J. Hamer. Peer-review in practice: eight years of aropä. *Assessment & Evaluation in Higher Education*, 43(7):1146–1165, 2018. doi: 10.1080/02602938.2018.1435776. URL https://doi.org/10.1080/02602938.2018.1435776.

K. Reily, P. Finnerty, and L. Terveen. Two peers are better than one: Aggregating peer reviews for computing assignments is surprisingly accurate. pages 115–124, 01 2009. doi: 10.1145/1531674.1531692.

J. Sauro. Measuring usability with the system usability scale (sus), 2011. URL https://measuringu.com/sus/. Online; accessed 18 June 2021.

X. Song, S. C. Goldstein, and M. Sakr. Using peer code review as an educational tool. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '20, page 173–179, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368742. doi: 10.1145/3341525.3387370. URL https://doi.org/10.1145/3341525.3387370.

Spring. Spring framework, 2021. URL https://spring.io/. Online; accessed 15 June 2021.

T. Storer. Cs4015 professional software development - week 5, requirements management, 2019. Lecture notes, Viewed 12 June 2021.

S. Stromeyer. 4 top reasons why good software architecture is essential for your success, Feb 2017. URL https://www.becompany.ch/en/blog/2017/02/14/why-good-architecture-counts. Online; accessed 16 June 2021.

S. Turner, M. A. Pérez-Quiñones, S. Edwards, and J. Chase. Student attitudes and motivation for peer review in cs2. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 347–352, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450305006. doi: 10.1145/1953163.1953268. URL https://doi.org/10.1145/1953163.1953268.

J. Wallen. What is ci/cd?, Oct 2019. URL https://www.techrepublic.com/article/what-is-cicd/. Online; accessed 20 June 2021.

Y. Wang, L. Yijun, M. Collins, and P. LIU. Process improvement of peer code review and behavior analysis of its participants. *SIGCSE Bull.*, 40(1):107–111, Mar. 2008. ISSN 0097-8418. doi: 10.1145/1352322.1352171. URL https://doi.org/10.1145/1352322.1352171.

Y. Wang, H. Li, Y. Feng, Y. Jiang, and Y. Liu. Assessment of programming language learning based on peer code review model: Implementation and experience report. *Computers & Education*, 59(2):412–422, 2012. ISSN 0360-1315. doi: https://doi.org/10.1016/j.compedu.2012.01.007. URL https://www.sciencedirect.com/science/article/pii/S0360131512000085.