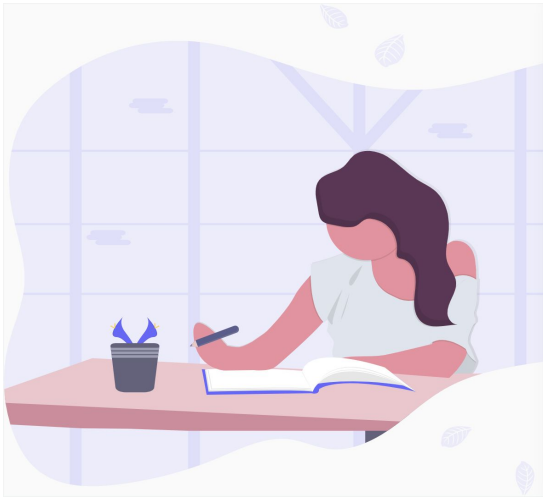


Delivery 4 Report

Team: Daddy Long Legs

Richard Yang yangud@bc.edu
Donghyuk (David) Kim kimbyi@bc.edu
Alexander Xu xubk@bc.edu
Seunghyun (Michael) Nam namsg@bc.edu

1. Abstract

In delivery 4, our team, Daddy Long Legs, completed the peer assessment while building off the class chosen archetype created by team “Smiley Face Logo”. Our development process was categorized into bins of responsibilities. First group members had to establish proficiency in React, which was the front end framework team Smiley Face logo used. In addition to understanding the model, our group had to architect a usable relational database system in order to store and configure usable data. Applying our knowledge in the infrastructure and takeaways from our proposed database system, we were then able to establish a system of communication optimized for our schedules and proficiencies, as well as prioritize which targetable functional requirements to satisfy so we could focus our efforts toward maximizing what we could accomplish in the limited/entropic time frame.

2. General Requirements

Here is a quick table that visually shows which requirements were fully done, assigned as an admin’s responsibility, semi-completed, and not completed. Afterwards, the completion of each requirement is explained.

	Fully Completed	**Admin Responsibility**	**Semi-Completed**	**Not Completed**
Requirements	1, 2, 3, 4, 5, 8,14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, Peer assessment reqs, 29, 30, 31	6, 7, 9	11, 12	10, 13, 25

Our software allows for the system administrator to create accounts for instructors, TA's and students. Additionally, the breadth of information required by the system is exactly up to the expectations of this assignment. Instructors, TAs, and students need to provide their full names, emails, and eagle id's in order for the system to randomly generate user passwords (req. 1).

In order to access the software, all users are greeted by an appropriate login page that requires the email and associated password for the individual, with each login page specified for a given type of user (req. 2). When a user attempts to make an account with an email or eagle id already stored in our relational database, our backend support would catch this match and prevent the user from creating the user (req. 3). Of course, all of this could only be implemented with the consideration of the three types of users, being admin, instructor, and student (req. 4).

Each course is saved in our database with three features: name, unique code, section number, year and semester of realization (req. 5). Classes are created by the admin upon a request by the teacher, which was a design choice to consolidate control within the role most suited for the task. While our system doesn't autonomously ensure a course has to have at least one instructor, we have the admins create the classes to ensure no class is made without an instructor (req. 6). This is because the interfaces provided by team "Smiley Face Logo" weren't conducive for allowing the system to do this automatically. While a course must have at least one, we've also added columns to our relational data table of courses to handle and store more than one professor for a course through our network of primary keys and architecture (req. 8). Our system however doesn't mandate that the combination of code, section number, year and semester of course must be unique, we leave that responsibility to admins (req. 7).

We also put student creation in the responsibility of the admin, therefore the system doesn't automatically handle this requirement (req. 9). This once again is due to the interface provided not being suitable to handle this within the system itself without admin intervention.

Our system is not capable of reading a file that has the information related to the course and to the teams (req. 10). In the teacher's "Teams and Students" tab, the instructor is able to see a list of all the teams with the team members' names underneath across all courses. The instructor is also able to observe all students by individual students. However, the instructor cannot change the student's team so this requirement is mostly but not fully completely satisfied (req. 11). Each team is required to have a team name which is mandated by the system, but we have to rely on the admin to enforce that each team has to have at least 2 students, so this requirement is mostly but not fully accomplished. (req. 12). We do not log the history of the team that a student belongs to mainly because we don't allow instructors to change student's teams in the first place so this requirement held no purpose anymore (req. 13).

Our system enables peer assessments to have 0 or more peer assessments (req. 14). This means a course can have no existing peer assessments and still be a valid course. In order to create a peer assessment the instructor inputs a name, start date, and due date (req. 15). Students will be reminded with an email containing the link to the home page on the arrival of the start date (req. 16). The system also automates the calculation of empirical results after each peer assessment ends for each team and student (req. 17).

Teachers are given full autonomy in seeing aggregated results, at both the team and student level by clicking a closed assessment in the class assessment tab (req. 18). Likewise, on the very same page instructors have the option to download results as a .csv file (req. 19). While

this function is not open for admins yet, this could be implemented. The issue is that this would require additional interfaces for the admin that weren't provided. Instructor's are further able to see who answered the peer assessment and who did not, and they also have the option to send a personal reminder through our system email (req. 20).

If a student doesn't answer the peer assessment, his/her peer assessment grade average will automatically be zero, overriding the grades his/her peers gave him/her (req. 21). When the instructor wants to, they can click the release results button to allow students to see the aggregated results of the peer assessment they received (req. 22). These results will only be averages without detail.

3. Instructor Landing Page Requirements

Each instructor's landing page shows a list of classes they teach (req. 23). Instructors can click the course they want to visit and they'll see all assessments associated with the course they clicked on (req. 24).

4. Student Landing Page Requirements

Students landing pages display the classes they're in similarly to the professor's landing page. This means that requirement 25 isn't satisfied. The reason for this is because given the other requirements, this requirement didn't make sense and would make the system much more confusing to use if we implemented this requirement. We believe our version of the user homepage is much simpler and better (req. 25). When results are released, students can easily navigate them by clicking the class they want to see, then going to the results tab and opening the assessment to see their aggregated results for each question (req. 26).

5. All User Requirements

All users of the system are allowed to change their passwords via the change password button on the navigation bar (req. 27). In order to confirm the change, we check user authenticity by requiring the confirmation of the current password (req. 28).

6. Peer Assessment Requirements

Each of the required questions are available for peer assessment creation (peer assessment requirement). We also enable the professor to create a new question (indicated in rubric). We are able to do this by creating a new instance of the question in our database each time it's deployed, allowing each instance to store individual responses. This allows for the protection of historical responses. The system displays the status of peer assessments for all users as open/closed in addition to their start and end dates (req. 29). Instructors are also allowed to update the deadlines of open peer assessments without losing information on the assessments (like grades) that were already stored (req. 30). Finally, the admin is allowed to add or remove instructors before opening the peer assessment to students (req. 31)

7. Methodology/Team Dynamic

Our approach to completing this delivery mostly followed the *democratic* team dynamic. While this was not intentional at first, we naturally gravitated towards this dynamic largely due to the early start we enacted on the project and the small group size. Due to these contexts, the

downsides of complex communication logistics found in large democratic groups and prolonged decision making are offsetted. It also helped that the scopes of this project were defined clearly, allowing us to delegate on the intermediary deliverables and not require the constant guidance from the professor, which could cause problems due to the lack of a "leader."

Sure enough, we were able to capitalize on finding errors early. For example, we were able to run through 3 different iterations of our database model structure within a week, including each of their implementations. As a result, our team felt tight knit due to the shared sense of being part of the solution.

In order to enjoy the full benefits of a democratic approach, we had to have consistent and frequent meeting times to maintain the inclusive feel among team members. This was hard since one of our teammates was stationed in South Korea, confronting us with a time zone road block. In response we scheduled our meetings largely at night to accommodate. Our scheduling of meetups evolved as we progressed with the project. For the early stages of the project, assessing our strengths and expectations of the deliverable pushed us to meet every other day. As our achievable goals, intermediary deliverables, and strengths became clearer, we were able to prolong the amount of time between zoom meetings to about every 3 days. This allowed us time to finish our delegated action items and have topics to discuss at our next meeting.

While we had a schedule for formal meetings, we had complete flexibility for informal ones. These were meetings that anyone in the group could call in order to address any roadblocks they may have. Informal meetings were sporadic, but served its purpose of enabling individuals in accomplishing their parts.

Throughout the project, informal roles were also established in the form of authorities. These authorities were dependent on team member strengths, delegated action items, and availability. For example, Richard was stronger in coding than the rest of the team and became an authority who the rest of the team would ask for delegation of tasks and coding questions. Alex and David took database systems and handled a large part of the database handling and became the authorities in that arena. Michael, due to his timezone circumstances, handled a lot of the unit and system testing. While we all contributed to each other's sphere of authority, when a question arose we all knew who to go to for guidance.

8. Database Model

In order to accommodate for our functional requirements, we had to implement a relational database network that would store information and allow us to deviate away from hardcoding features and information. Below are the different different data tables stored within our database system, with a brief explanation of their features.

User : Stores user and their related information

eagle_id = Unique user id
first_name = Stores user first name as a charField
last_name = Stores last name as a charField
email = Stores user email as a charField
password = Stores user password as a charField
type = A binary field that denotes if user is a instructor or not

Class: Stores class and its information

class_name = Stores name of class as a charField
section = Stores section information

semester = Stores which semester the class is offered
year = Stores year offered
code = Stores the class code

Group: Stores group and its related information

group_name = Stores group name as a charField
class_id = Stores associated class id

Group_Student: Intermediary data table to hand the "many to many" relationship between Student and Group

student_id = foreign key to user data table
group_id = foreign key to group data table

Class_Professors: Intermediary data table to hand the "many to many" relationship between Class and Professor

professor_id = foreign key to in user data table
class_id = A foreign key to class data table

Assessment: Stores assessment and its related information

assessment_name = Stores assessment name as charField
creation_date = Logs date assessment was created
start_date = Stores inputted start date
due_date = Stores inputted due date
class_id = A foreign key to class data table
released = A boolean field that indicates if scores has been sent
sent_email = A boolean field that indicates if email has been sent

Question: Stores questions and its related information

question =Stores question name as charField
type = A categorical field that indicates what kind of question it is (MC, Open Response)

Assessment_Question: Intermediary data table to hand the "many to many" relationship between Assessment and Question

assessment_id = A foreign key to assessment data table

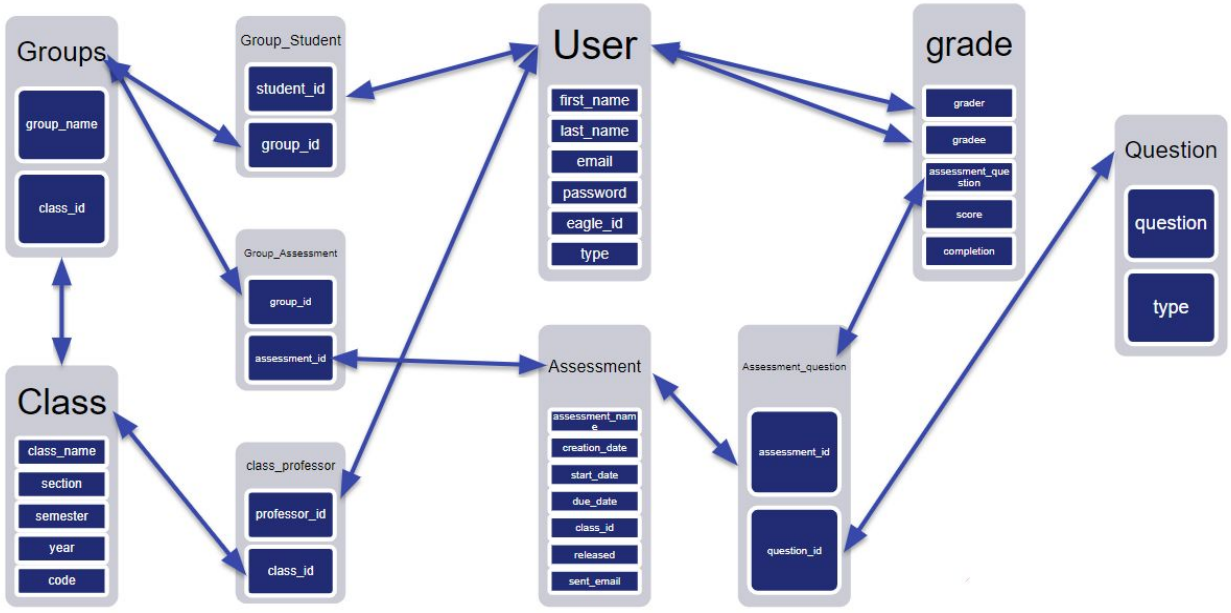
question_id = A foreign key to question data table

Group_Assessment: Intermediary data table to hand the “many to many” relationship between Group and Assessment

group_id = A foreign key to group data table
assessment_id = A foreign key to assessment data table

Grade: Stores the grade and related information

grader = A foreign key to user data table
gradee = A foreign key to user data table
assessment_question = A foreign key to assessment data table
score = Stores the score of the assessment question
completion = A boolean field that indicates if the assessment question is completed.



9. Database Model Traceability With Interfaces

Below in a table are our interfaces and functionalities with the tables that are specifically queried for in the database. To see how each is used, see the backend source code. Many interfaces use many tables, sometimes querying one table for info to be used in another query.

Interface Name	**Tables Accessed**
Login	User
Change Password	User
Professor Landing Page	User, Class_Professor, Class
Student Landing Page	User, Group_Student, Group, Class
Professor Peer Assessment Page	Class, Assessment
Create Assessment Functionality	Class, Assessment, Group_Assessment, Assessment_Question, Grade
Deadline Update	Assessment
Student Assessment Page	User, Grade, Class, Assessment, Group_Student, Question
Student Peer Assessment Submit	User, Assessment, Question, Asessment_Question, Grade
Student Result Page	Question, User, Grade, Assessment_Question, Assessment, Class
Student Aggregated Result	User, Assessment, Grade, Question, Assessment_Question
Question Page	Question
Add Question Button	Question
Professor Result Page	Question, Assessment_Question, Class, Group, Group_Student, User, Grade
Release Results Button	Assessment
Download Results	Assessment, Group_Assessment, Group_Student, Assessment_Question, Grade
Professor Student Detailed Results Modal	User, Question, Assessment_Question, Grade,
Professor Team Detailed Results Modal	Assessment, Group, Group_Student, Assessment_Question, Grade
Remind Student Email	Assessment, User
View Teams and Students Page	Class, Group, Group_Student, User
Send start date email	Assessment, Group_Assessment, Group_Student

Several interfaces like the login change password, question, and some assessment interfaces only require accessing one table, but others require much more. Functionalities dealing with groups and assessments either require multiple sets of information, or a chain of information to accomplish the task. For example, interfaces like peer assessment grading and results require multiple tables because of the many foreign keys relationships. In order to get individual grade results, one needs to query a user, the group_student's table for the students team members, the assessment, and the assessment_questions all to get one single grade entry. While this may seem like many tables to get information, this data model is very stable, dynamic, accommodating, and consistent with how to query the right data. With the table above, each interface has their corresponding model traceability for data.