## **OSS Projects on Expertiza for Spring 2024**

This project list consists of project descriptions aimed at refactoring the current version of Expertiza and also descriptions aimed at reimplementing a new version of Expertiza. The current version is an MVC in Ruby on Rails and the new version has an API written in Ruby on Rails and a front end application written in React and Typescript.

## On all projects, make sure you follow these steps:

1. Establish your project Github Repository and Accompanying [Github Project Board](https://docs.google.com/document/d/1NWSI2CP1S4QMEL3KH4bupKXcqmjN30R0pEFVqUSBJMY/edit?usp=sharing).
   1. All groups are required to plan and track project tasks using Project Boards. This will also be used as a mechanism to monitor teammate contributions and management of your project.
2. **Create a pull request as soon as you make your first commit so that you have a much higher chance to get a good score** and the staff can track your progress and offer help where needed.
3. Make sure to sync with the **main** branch frequently.
4. If your project is focused on fixing multiple issues, it will help the Expertiza team if you create separate pull requests, one for each issue you have fixed.
5. For each project, you **must** write related tests.
6. [Common mistakes checklist](https://docs.google.com/a/ncsu.edu/document/d/1s5ucr6N2EWU8jH60UdmopHdx4otQ4l9S2gfl5PmcPvY/edit?usp=sharing) (we will check these common mistakes during grading.)
7. Some good RSpec tutorials. [RSpec tutorial slides](https://docs.google.com/presentation/d/1fSgEPdFAm13vYbwf-VimjUFxzQqs-lL2_6S7ZqbuuQM/edit?usp=drivesdk), [video (until 9:32)](https://youtu.be/dzkVfaKChSU?t=35s), <http://rspec.info> and <http://www.betterspecs.org>.

## **Expertiza repositories on Github**

## For projects working with the existing code base, including refactoring projects: <https://github.com/expertiza/expertiza>

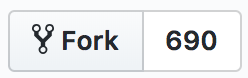
* For front-end reimplementation projects: <https://github.com/expertiza/reimplementation-front-end>
* For back-end reimplementation projects: <https://github.com/expertiza/reimplementation-back-end>

## For refactoring projects

1. You may find that some functions are already broken before you do your refactoring. It is not acceptable if you take the broken code, refactor it so that it breaks in the same way as before. Instead you should try to fix the code to make it work first. You are also welcome to talk to the Expertiza Support team ([expertiza-support@lists.ncsu.edu](mailto:expertiza-support@lists.ncsu.edu))
2. When you finish your project, synchronize it with the main branch and resolve conflicts. Make sure that all tests succeed, as evidenced by Git Actions right after submitting the pull request.
3. When you create the test cases, keep in mind that the testing DB does not have the production data in it. If you need some pre-existing entities like courses, assignments or students, you can find them in factories folder.

Fork the appropriate Expertiza repo into your Github account.

**To fork the project,** click on the button that looks like this:



After git clone your forked repository, you can execute the command **run cd expertiza && bash ./setup.sh** to install dependencies.

**Please view the** [**Expertiza YouTube channel**](https://www.youtube.com/channel/UCdKXzox7hrWjfOMML6FzTWg) **before beginning your project.** The videos on this channel will show you many areas of the system that you do not see as a student, but which are important in setting up and managing courses and assignments in Expertiza.

**Details of the Expertiza project, including links to documentation and development may be found at:** <http://research.csc.ncsu.edu/efg/expertiza> [if the link does not work, see note below on wiki links][**.** Especially note the Documentation on Database Tables,](http://wikis.lib.ncsu.edu/index.php/Documentation_on_Database_Tables) which, though not completely up to date, explains almost everything you would want to know about Expertiza db tables.

## **Setting up Expertiza environment (see** [**Google doc**](https://docs.google.com/document/d/1hfNDT8hPmGf90FijsnWz6cx0SYUyraRxHfgrT8UVbR8/edit)**)**

## For reimplementation projects

For the reimplementation of the Expertiza application, we will use Rails API instead of a Rails application. The main difference between the two is that Rails API is a slimmed-down version of Rails that is specifically designed to create APIs. Since we are planning on having separate applications for the front end and back end, we will be using Rails APIs.

The new front end for Expertiza is implemented in Typescript. To familiarize yourself with Typescript, first, read the [Typescript teaching notes](https://docs.google.com/document/d/1IrKIiP3i0vphW435KaS66ofZ53qlIkbD_68Bz0hhkU4/edit#heading=h.wimu45po0rl5). Then walk through the implementation of the [Typescript AmazeZone](https://docs.google.com/document/d/1_CyV_VotcVPGpGpLWOYHuqhfCrnxcJH4URQRQuQS55o/edit#heading=h.d1owl0c3qyx5).

Please follow the structure created for `Roles, Users, and Assignments` in this [repository](https://github.com/expertiza/reimplementation-back-end), and follow the Role model and Roles controller. Also, take a look at the routes on how to define your resources.

Create your controllers in → *controller/api/v1*

If your project is a front-end project, the code should be written in TypeScript. First, review [these instructions](https://docs.google.com/document/d/1IrKIiP3i0vphW435KaS66ofZ53qlIkbD_68Bz0hhkU4/edit#heading=h.wimu45po0rl5), and then try doing the [AmazeZone exercise in React and Typescript](https://docs.google.com/document/d/1_CyV_VotcVPGpGpLWOYHuqhfCrnxcJH4URQRQuQS55o/edit#heading=h.d1owl0c3qyx5). Then work with your mentor to design your user interface.

**Setting up a reimplementation environment. (see** [**Google doc**](https://docs.google.com/document/d/1M0FRp_LqrkQ9E03rIqGM30PpP9UyOpmBEvS0seIWBGw/edit?usp=sharing)**)**

**Guidelines for reimplementation projects**. (see [Google Docs](https://docs.google.com/document/d/1aOZsZd8cweT8lFcUcJ028OcT_qmGoNYYkbifVkZV_Yk/edit))

## **What to submit**

* On Wednesday, March 20, the project is due. You should submit to Expertiza …
  + your pull request (which should have been created when you started the project),
  + a wiki page talking about what you have already done
    - Here are examples of good write ups for the OSS projects: [**E1553**](http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2015/oss_E1553_AAJ), [**E1566**](http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2015/oss_E1566_ARB),[**E1571**](http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2015/oss_E1571), [**M1504**](http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2015/oss_M1504_JJD), [**M1506**](http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2015/Mozilla_Refactor_GLES2)
  + for testing projects, record a video of tests running and submit a link
  + r other projects, deploy your application on Heroku or VCL and make sure the server is up for 3 weeks
  + A link to your Github Repo, which should also grant access to your Github Project Board.

If a link to a wiki page does not work, in the URL please replace “wiki.expertiza” with “old-wiki.expertiza.csc”. This will be necessary until we finish moving the wiki to the new server.

## **E2400. Allow reviewers to bid on what to review**

**Mentor:** Ed Gehringer (efg@ncsu.edu)

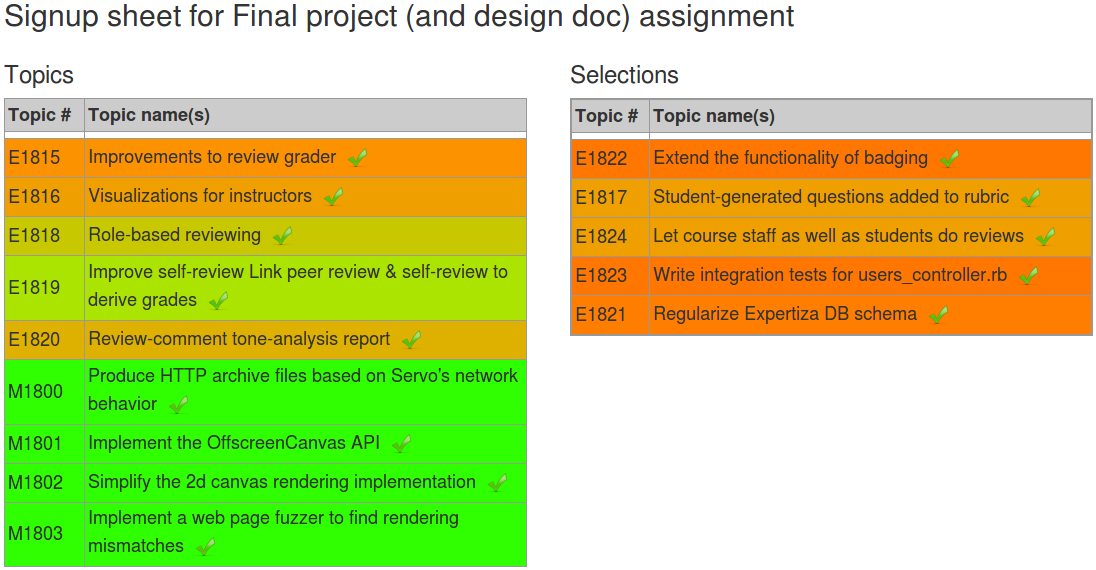
**What it does:** Assigning reviews to users is a complicated process. Currently, as per previous year’s implementation on Expertiza’s beta branch, reviews are assigned using the bidding algorithm. Although, the functionality works, but has some bugs which needs to be fixed before we can take it to production.

In my judgment, it would be easier to go back to the original project, [E2151](https://docs.google.com/document/d/1slx4HPIbgTH-psIKMSCF-HDF9brxf-FuYhzVT9ZiIrM/edit#heading=h.dcffq96f1b7t) ([pull request](https://github.com/expertiza/expertiza/pull/2150)), and try to implement its features. (To see what bidding for topics looks like, try [this page](https://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2016/E1631._Refactoring_Bidding_Interface).)

**What needs to be done:**

* action\_allowed needs to use the [authentication utilities](https://wiki.expertiza.ncsu.edu/index.php?title=E1915_Authorization_Utilities).
* Review\_bids\_others\_work is a DRY violation
* Run\_bidding algorithm should be assign\_reviewers
* Currently it doesn't work if some student does not bid. In this case, algorithm needs to be fixed to ignore anyone who didn’t bid, and let them choose from what’s left over.
* Make sure your code works on individual assignments, i.e., assignments where participants sign up for topics instead of teams. In this case, a team is created for each participant when they sign up. So the code *should* work for assignments to which either individuals or teams submit.
* To make the functionality more intuitive, include a message to say how many students are eligible to submit bids, how many have submitted their bids, when deadline for submitting bids is.
* ~~Ability to allow the algorithm to run again after running once.~~
* Why are the methods in review\_bid.rb class methods? Can we change them to instance methods or move it to helpers?
* In the previous implementation wiki, there are edge cases which are not exhaustively tested. Should test those edge cases thoroughly and add more edge case testing
* Test whether the topic changes color depending on the number of outstanding bids

Bidding UI:



**Important files:**

* [Top trading cycles algorithm](https://www.jstor.org/stable/pdf/3132114.pdf?casa_token=Bp8Syf8Q0yYAAAAA:f7zDJdWurp5cYlHtSHaUxRbDc1o4sL0Qx8UTnk7C5eeXPhgyiTOaWqgftX-nNJ48s6SGyPBzH3_U3Yv6Pfapo1OJaj-estvNyRl60LNQi5QgGW3LmAW8pQ)
* [Peerlogic](https://github.com/peerlogic/IntelligentAssignment/blob/master/app/app.py)
* [Lottery Controller](https://github.com/expertiza/expertiza/blob/master/app/controllers/lottery_controller.rb)
* [Review Mapping Controller](https://github.com/expertiza/expertiza/blob/master/app/controllers/review_mapping_controller.rb)

**Latest version**: The project was done in [Spring 2022](https://github.com/expertiza/expertiza/pull/2382). Fall 2021 links:

* <https://github.com/expertiza/expertiza/pull/2150>
* <https://www.youtube.com/watch?v=wxXALQNWFcc>
* <https://expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Fall_2021_-_E2151._Allow_reviewers_to_bid_on_what_to_review>

## **E2401. Implementing and testing import & export controllers**

**Mentor:** Ed Gehringer (efg@ncsu.edu)

**What is needed:** In Expertiza, many kinds of files can be imported or exported: lists of users for whom accounts are to be created, lists of teams that have been created or need to be created, lists of topics that users are to be able to sign up for, or instructor or peer scores that have been given for a particular assignment. Historically, all of these import and export methods were written individually, using similar code patterns.

But then one day it was noticed that there was a very simple way to specify import and export functions: Give the name of a db table, and list the fields of the table that needed to be imported or exported, together with any filters that would determine which records of the table needed to be exported. Once a controller was written to import (or export) arbitrary tables, then all of the existing import and export controllers could be replaced by a few lines of code that would just invoke the new import and export controllers. This was implemented in Project E1923 ([project specification](https://docs.google.com/document/d/1Ozw2Bj2u_LyeKUdDqAyQtz1stEIKrxkmRnwmNyUgmzw/edit#heading=h.gt591iba2jb1), [project description](https://wiki.expertiza.ncsu.edu/index.php?title=CSC517_Spring_2019/E1923_New_Framework_For_Import/Export), [pull request](https://github.com/expertiza/expertiza/pull/1438), [video](https://youtu.be/1nSnjta_zIc)).

Your task is to take existing import and export code and modify it to use the new controllers, then to write comprehensive tests for the new import and export controllers.

**What might you be interested in importing or exporting?** Imports

* **participants**, for an assignment or a course
* questions, as part of a rubric (the rubric would be what you wanted to import; it would consist of a **questionnaire** object as well as a set of questions).
* **sign\_up\_topics** (signup\_topics), which are like the topics you signed up for in Program 3
* signed\_up\_teams (you would import a list of teams that have specific topics)
* **teams**; these may be assignment teams or course teams
* **users**; to create new users

Exports

* **grade\_for\_submission**, for all teams in an assignment, along with user IDs
* sign\_up\_topics (you might want to use the same topics in another assignment)
* teams
* answer\_tags for a particular assignment

**Previous implementation**

* <https://expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2022_-_E2238._Implementing_and_testing_Import_%26_Export_controllers>
* <https://github.com/expertiza/expertiza/pull/2396>
* <https://drive.google.com/file/d/1obg-vu1lSX28kILlATQkyTFkGGZXnoVq/view?usp=sharing>

**Our comments on the implementation:** The team has used the new framework for import and export. They prioritized the specific models as there were initially a lot of models to add import/export. Schema.rb is skewing the pull request data due to probably rails migrate commands. Currently, the Travis CI pipeline has tests failing.

This project was not merged due to failed tests. Please assure that those tests pass, and also implement import and export for at least one additional class (the mentor can help you identify one).

## **E2402. User management and users table**

**Mentor:** Ed Gehringer (efg@ncsu.edu)

**What is needed:** In the Expertiza code, user-ids are called “names” and user’s names are called “fullnames”. This is an artifact of the Goldberg gem that we used long ago to implement authentication and menus. It is confusing to users, and long overdue for elimination. But these terms are used in many places in the code and the user interface, and it’s not an easy job to change them consistently. The term “user” occurs at least 892 times, and “name” occurs over 1000 times in the Expertiza code. The first step is to make sure that a user ID is called “user-id” and users’ names are called “names” everywhere in the code and UI, but be careful not to change these terms if they mean anything other than an attribute of a User object.

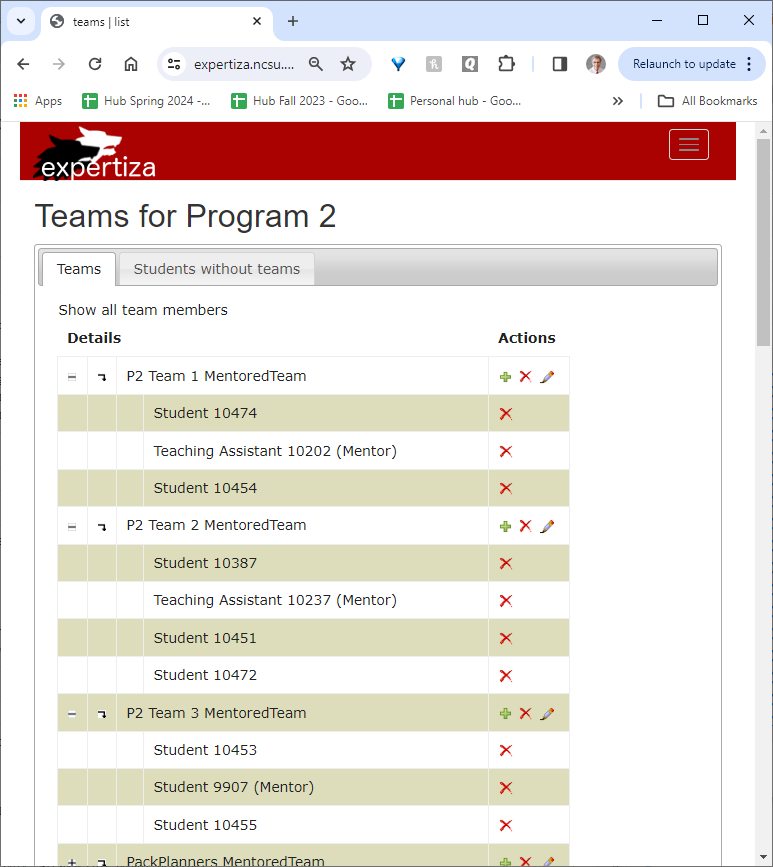
The next issue is that there is currently no way to display all of the users on the same screen. One hundred users are displayed at a time, and there is no way to change this. There should be a way to display 25, 50, or 100 users per page, or just to display all the users on a single page. This would be slow, but it would facilitate text searches for names, user-ids, roles, and everything else that shows up in a listing of users.

## **E2403. Mentor-meeting management**

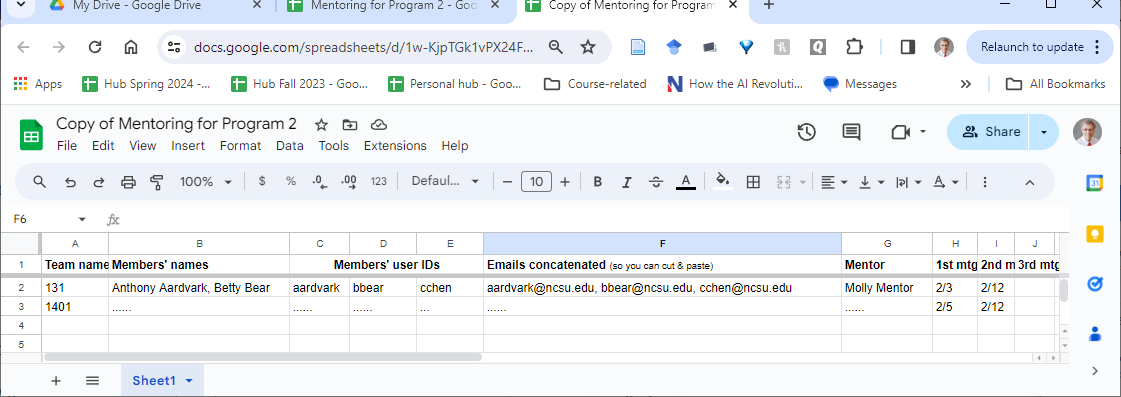
**Mentor:** Ed Gehringer (efg@ncsu.edu)

**Background:** This semester was the first time that we used automatic mentor assignment for Program 2. Before it was functional, somebody (me) would have to look at the teams that were forming and manually assign mentors. So it was a great relief when code was finally operational and I didn’t have to check the team list multiple times a day to see if new mentors needed to be assigned.

But, not so quick! Yes, it was much easier for mentors to be *assigned* to teams, but teams and mentors need to be told (via email) when these assignments were made. The code to do this is missing or not working. It needs to be made operational. Also, the only listing of teams is on the “Create Teams” page for the assignment. It shows each team member on a separate line, with the mentor in an arbitrary place on the team. Here is a sample listing, using Expertiza’s anonymized view:



What we’d like is a listing that looks something like this, sortable by team name, mentor name, or meeting date:



Expertiza doesn’t know when mentors met with teams, so there would have to be text fields where this information could be entered. The page would have to be accessible to all mentors, but a mentor could only enter meetings for the teams they mentored (except that the instructor could edit any of the meeting dates for any mentor).

As a possible extension, maybe for a final project, it would be good to have the same functionality for the OSS project assignment, where the mentor is associated with the project topic. Doing this well would require quite a bit of attention to the design.

Example of email sent by Expertiza: conjured up by notify\_grade\_conflict\_message(defn) in mailer.rb

[expertizamailer@gmail.com](mailto:expertizamailer@gmail.com) Feb 18, 2024, 11:58 AM (12 days ago)

Instructor of Program 2,

Person 1 has just submitted a review of Person 2's artifact.The new score is 17.50%. This score differs from the previous average score by more than the threshold. The url for conflicting response url is https://expertiza.ncsu.edu/response/view?id=199999.

The url for summary of responses is https://expertiza.ncsu.edu/grades/view\_team?id=51999.

If you want to change the notification threshold on grade conflicts, you can find it on "Rubrics" tab on assignment edit page: https://expertiza.ncsu.edu/assignments/1994/edit.

Assignment: Program 2

mail may be a Rails library method or a method of the Sidekiq

What are some times that Expertiza sends mail?

* When a review is posted on someone’s work, the reviewee gets email.
* When an author resubmits work, the reviewers get emailed.
* When someone requests a new password, it gets emailed to them.
* The instructor gets notified when different reviewers submit dissimilar scores for a submission.
* I think you get emailed when someone invites you to join a team.

This seems to be the documentation on the [initial bidding project](https://docs.google.com/document/d/1XKplosor8nAiv64kN5T06UhsjpHHWFcoPKvgHRjilTA/edit#heading=h.5cgh9aixuut7).

## E2404. Refactor student\_teams functionality

**Mentor:** [Ananya Mantravadi](mailto:amantra@ncsu.edu)

**Background:**

* Assignment:First OSS Project you just finished was an assignment.
* Users & Participants:Everyone who took the OODD class is a user in expertiza, and instructors can add users as participants in an assignment so that they can access it.
* Almost everything within Assignments is done by participants, not users. That is, everything except **teams\_users**. In the [documentation](https://wiki.expertiza.ncsu.edu/index.php?title=Teams_users)**,** it can be noted that the team\_users table references users instead of participants.
* This anomaly causes problems with how student teams are rendered in the UI, and it doesn’t go well with the new functionality that was recently introduced.

**What is needed:**   
Before you begin reading what is needed, please have a look at the documentation of [teams\_users](https://wiki.expertiza.ncsu.edu/index.php?title=Teams_users) table. The last team did a really good job with this functionality, but due to some conflicts with parallel developments, could not be merged. Please refer to their wiki [here](https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Spring_2022_-_E2243._Refactor_student_teams_functionality), and [here](https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Spring_2023_-_E2304_Refactor_student_teams_functionality). **It is highly recommended to meet with your mentors before you start working on this project.**

* Changing the schema: Please don’t directly modify schema.rb file. Any changes in schema has to be done through the migration files.
  + We need to rename the table table teams\_users to teams\_participants
  + Create a column that would be called “participant\_id”, which will be a foreign key that references [participants](https://wiki.expertiza.ncsu.edu/index.php?title=Participants) table.
  + Create a migration that fetches the assignment\_id for each tuple from the [teams](https://wiki.expertiza.ncsu.edu/index.php?title=Teams) table. We would find the participant with the help of the assignment\_id we just got from the teams table and the user\_id that is already present in the current table, and store its “id” in the participant\_id column we created in the last step. You have to come with optimal migration strategy that reliably calculates the participant\_id
  + Find out all the places where user\_id is being used, refactor those so that participant id is used. Hint: It would be good if changes are made in one place so that you won’t have to change code everywhere
  + The major part is done here!
* Refactoring the teams rendering (/views/student\_teams/view.html.erb)
  + The logic in the team's view (where participants can see their team members) that iterates and fetches information of all the team members and displays them would be refactored so that it matches our new design.
  + [A data migration will be needed.]
  + The view is quite bloated and has lots of functionalities written on it. We’ll be making partials and placing those functionality in their specific partials.
  + The new changes should not break the view that works for data where things are populated with user\_id. What I suggest is separate the older functionality into a separate partial and code the new functionality into a separate partial.
  + That being said, you must ensure that going forward, all the teams are being formed with participant\_id instead of user\_id.
* Update logic for all the crud functionalities in teams\_users controller (wherever applicable).
* Rename the teams\_users\_controller to teams\_participants\_controller (and all its references)
* There is some code where sql queries are written to fetch data (Line 16 & 17 on app/models/signed\_up\_team.rb), which should be replaced with fetching data from rails ActiveRecord methods.
* The html.erb files have extensive ruby code written for fetching data from the database. This code should be removed from the view files and placed into helper classes.
* Extensive test cases should be present for all the code that is written. This will be the major contributor to your grades.
* There’s a bug in the Team information page (“Your team” tab where you can view and manage your team), which allows the same user to be added twice (or many times). Below image can help you understand the issue better:  
  This is most likely due to the instructor adding a member to a team whom you already invited, but you’ve to dig deep into how exactly you can recreate this issue. Do mention in your writeup all the ways you can create a team (this will help us in weeding out redundant features). Bonus 10 points if you can do all that (:

Reference Links:

1. E2147 role based reviewing project: <https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2021_-_E2147._Role-based_reviewing>

## E2405. Refactor review\_mapping\_helper.rb

**Mentor:** [Ananya Mantravadi](mailto:amantra@ncsu.edu)

**What is wrong:**

The review\_mapping\_helper.rb has methods that exceed the limit on lines of code Also, it is missing proper comments for each functionality. The cyclomatic complexity of most of the methods is way too high as per the standard defined in the Code Climate.

**What needs to be done:**

1. Use Code Climate to do a diagnosis for Expertiza. Search for the issues involving the files mentioned above.
2. Read about cyclomatic complexity [here](https://en.wikipedia.org/wiki/Cyclomatic_complexity).
3. Fix all the methods having the issue with cyclomatic complexity, assignment branch condition size, extra commas and space or lack of it, and assigning branching condition.
4. Write test cases of any new code written. Test the existing functionality after refactoring

**Previous versions of project**

* Spring 2023:
  + <https://github.com/expertiza/expertiza/pull/2550>
  + <https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Spring_2023_-_E2301._Refactor_review_maping_helper>
  + <https://github.com/ritwik4690/expertiza>
* Fall 2019:
  + <http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2019_-_E1948._Refactor_review_mapping_helper.rb>
  + <https://github.com/expertiza/expertiza/pull/1526>
  + <https://bit.ly/2NiWlgI>
  + <https://github.com/ArshdeepSinghSyal/expertiza/tree/beta>

**Comments on previous version**

**Pros:**

1. Good refactoring of code
2. Added comments in existing code which makes it easier to understand the code.
3. Resolved code climate issue – completed their task
4. Created methods doing only 1 functionality – good
5. Video showed the result of code climate which passed all the earlier failed test cases.

**Cons**:

1. Asked the team to create test cases for all the new methods created. No new test cases written.
2. Even though the overall method is tested by the original test cases, it is always mandatory to write test cases for the new methods as well.
3. They could have shown the working of the project as well in the video.

**Reviews**:

1. Appropriate variable and method name except for naming convention not followed in naming variable teamID. It should be team\_id
2. A few existing methods' names are too long, this could have been changed by the team if they were working on that file - get\_each\_review\_and\_feedback\_response\_map, get\_css\_style\_for\_calibration\_report
3. Test coverage decreased because they created new methods without writing test cases for them.

## E2406. Further refactoring and improvement of review\_mapping\_helper

**Mentor:** [Ananya Mantravadi](mailto:amantra@ncsu.edu)

What is needed: This project builds on E2301, and should begin with the refactoring done by that project. That project focused on simplifying the methods in review\_mapping\_helper, while this project looks at making the code more understandable and transparent.

* method get\_data\_for\_review\_report marshals a lot of data together and passes back a data structure. It is used in views/reports/\_review\_report.html.erb, but it is quite difficult to see how the data is being displayed. It violates the Expert pattern, because the view needs to know how to break apart the structure that is passed to it. Doing the same thing with partials in views/reports would make the code easier to follow
* The next three methods involve team “color”. Color-coding is explained on [this page](https://docs.google.com/document/d/1WYiXxhYyycp9a3I0GTC-4KFgHrg65Tf27ijJNQvxhmk/edit#heading=h.mo1zn1d9w38l) and [this page](https://wiki-expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Fall_2017/E1789_Semester_Project) for the E1789 project and [this page](https://wiki-expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2018-_Project_E1815:_Improvements_to_review_grader) for E1815. The code for these methods is not at all clear, and should be refactored. And please use the American spelling “color”.
* Various method names begin with get. This is not Ruby-like. Change the names to something more appropriate.
* get\_awarded\_review\_score computes an overall score based upon scores awarded in individual rounds. This is one of many places in Expertiza where scores are being calculated. [Score-calculation code](https://wiki-expertiza.csc.ncsu.edu/index.php/Scoring_%26_Grading_Methods_(Fall_%2721)) for multiple rounds is being standardized now, in response\_map.rb. Change this method to use the new code.
* The method sort\_reviewer\_by\_review\_volume\_desc should be generalized so that it can sort by any metric, not just review volume. Other metrics might include number of suggestions, or number of suggestions + number of problems detected. This method should not be counting the number of review rounds! Since other places in the code will need to know the number of review rounds, it should be calculated somewhere else in the system.
* The next several methods generate charts. They are cohesive enough that they should be in their own separate file, either another helper or a mixin.
* Then there is a method list\_review\_submissions. It’s not at all clear that this method is needed, though it is used in one view. Look on the Expertiza wiki and see if there is a better way.
* There are several methods for feedback\_response\_maps. It is not at all clear why they are here. Look on the Expertiza wiki for the documentation, and see if you can replace them by calls to other methods, or at least make it clearer what they are doing.
* The file ends with three small classes being defined. There are no comments at all to explain what is being done. Look them up on the Expertiza wiki and refactor or comment them, whichever seems more appropriate.

## **E2407. Refactor review\_mapping\_controller.rb**

**Mentor:** [Ananya Mantravadi](mailto:amantra@ncsu.edu)

**What is wrong with it:**

The review\_mapping\_controller is a long and complex file. Most of the methods are sparsely commented on. Some methods are way too long to understand, please break them down into pieces for better understanding. Also, the few instances of code duplication that exist should also be removed.

**What needs to be done:**

1. Refactor the long methods in review\_mapping\_controller.rb like assign\_reviewer\_dynamically, add\_reviewer, automatic\_review\_mapping, peer\_review\_strategy, etc.
2. Rename variable names to convey what they are actually used for.
3. Replace switch statements with subclasses methods
4. Create models for the subclasses
5. Remove hardcoded parameters
6. Add meaningful comments and edit/remove/do-not unnecessary comments.
7. Try to increase the test coverage

Provide extensive comments for all the code you view/refactor/modify. Ensure that code changes do not break any functionality. Refactoring method names *might cause cascaded updates in other files.*

Avoid adding new gems as they would break the build.

**Previous versions of the projects:**

* Fall 2023:
  + <https://github.com/expertiza/expertiza/pull/2657>
  + <https://github.com/amit-99/NCSU_OODD_expertiza/tree/main>
  + <https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2023_-_E2356._Refactor_review_mapping_controller.rb_(Phase_2)>
* Spring 2022:
  + <https://github.com/expertiza/expertiza/pull/2303>
  + <https://github.com/SriPallaviDamuluri/expertiza/projects/1>
  + [CSC/ECE 517 Spring 2022 - E2224: Refactor review mapping controller - PG\_Wiki (ncsu.edu)](https://old-wiki.expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2022_-_E2224:_Refactor_review_mapping_controller)

**Comments on previous versions**

E2224: This project is to refactor the long methods in review\_mapping\_controller.rb, rename variable names to convey what they are actually used for, replace switch statements with subclasses methods, create models for the subclasses, remove hardcoded parameters. This team achieved most of these goals, but a previous team has done similar work. This team does not provide enough new tests to the previous team work. More comments on the modified code are needed. Comment on E2224: Slated to be merged? Yes. Future work: move calibration code to another file, to the extent possible

E2356: The video looks fine, covering the code changes, bug fixes, and methods testing. Every method was tested using the test cases provided by Mustafa. The video demonstrates that the refactoring changes work, as shown by UI testing. The bugs fixed were assign reviewers, showing a walk back before saving the assignment, and the check for outstanding reviews if a participant has already completed the required reviews. The bug fixes can be explained more by adding comments. A few methods, such as check\_outstanding\_reviews, can be renamed. Also, some methods need better, valuable comments. Feedback on design doc: This is a very good description of the changes made, and the rationale for making them. However, some of the screenshots have very small text, not particularly readable. It would be better to use Github diffs to show the changes made. Another approach that might have helped would be to crop the screenshots on the right, and expand them vertically, so that the code would appear larger.

## E2408. Refactor course.rb and course\_team.rb models

**Mentor:** [Ameya Vaichalkar](mailto:agvaicha@ncsu.edu)

**Background:** course.rb and course\_team.rb models are objects containing information about the course and their associated teams respectively. CourseTeams are used when an instructor wants to use the same teams for the entire semester. CourseTeam is one subclass of Team ; AssignmentTeam is the other. An AssignmentTeam is a team that is created for students to do a single assignment. It is possible to copy CourseTeams to an assignment, where they become AssignmentTeams, or vice versa.

The Course and CourseTeam models have some code smells/issues which need to be fixed. There are a few methods in these models that shouldn’t be in those models, and a few methods that are no longer used. Some methods contain logic that is supposed to be in some other file. Below are descriptions of issues which you will be working on.

**Issues to be fixed in course.rb:**

* Check whether get\_participant is used anywhere in the codebase,if not ,then can we remove it?
* CourseTeam and Course both have add\_participants. Why? If one of them is not used, remove it. Should CourseTeam’s add\_participant be called add\_member? Should it be in the Team superclass?
* copy\_participants is used to copy assignment participants to a course; what would be a better name for this method?
* Methods that do not make sense in a model should be removed and added to the appropriate class/helper.

**Issues to be fixed in course\_team.rb:**

* AssignmentTeams and CourseTeams should be handled analogously.
* If there's an assignment\_id method, there should also be a course\_id method (that would be nil in AssignmentTeam). But if you don't need one of those two methods, why would you need the other one?
* Why is Course.find(id) ,it should be Course.find(course\_id)?
* Rename the copy method; it is copying a CourseTeam to an AssignmentTeam; what would be a better name for this method?
* Are import & export done the way they should be, according to Project E1923? Could import & export be moved to team.rb?
* If there is any dead code (not being used anymore ) delete it .
* Methods that do not make sense in a model should be removed and added to the appropriate class/helper.
* Caution should be given for the build to pass, as this is a super important component.I would suggest incremental commits, and checking if everything works.

## E2409. Refactor sign\_up\_sheet\_controller.rb

**Mentor:** [Anvitha Reddy Gutha](mailto:agutha@ncsu.edu) ([agutha@ncsu.edu](mailto:agutha@ncsu.edu))

**Files involved:** sign\_up\_sheet\_controller.rb

**What needs to be done:**

Provide extensive comments for all the code you refactor. Please do not change the database schema.

Ensure that any code changes do not break any functionality. Refactoring method names might cause cascaded updates in other files.

1. Naming is inconsistent. When used as a verb, “sign up” is two words. When an adjective or a noun, it should be one word, e.g., “signup\_sheet\_controller.” Cf. “sign\_up\_as\_instructor”. Please make the naming consistent. Of course, this will result in changes in calling methods as well. [Mostly fixed by previous version of project.]
2. Update method has a plethora of instance variables defined before updating. These might not be necessary (e.g., look at update method of bookmarks\_controller). Decide whether so many instance variables are really needed. Refactor the variables not needed out. [This may have already been fixed, but save\_topic\_deadline also has these problems.]
3. Add\_signup\_topics\_staggered does not do anything different from add\_signup\_topics. Separate functions are needed, because add\_signup\_topics\_staggered needs to make sure that the deadlines are set. [Assignment 1042 has staggered deadlines]
4. Several method names can be improved (including: load\_add\_signup\_topics, list, ad\_info etc.)
5. What are differences between signup\_as\_instructor and signup\_as\_instructor\_action methods? Investigate if they are needed and improve the method names if both are needed. Provide comments as to what each method does.
6. The list method is too long and is sparsely commented. Provide comments and identify if the method can be split or made cleaner by moving code to models or helper methods.
7. Refactor participants variable in load\_add\_signup\_topics [In retrospect, the meaning of this is not clear. The @participants variable is used in a way that is very obscure, with code spread across several views, and no comments saying what it is doing. It used to be that participants (individual students) signed up for topics. Now, only teams can sign up for topics. So @participants do not make sense.
8. Signup\_as\_instructor\_action has an if-else ladder. It can be made more elegant. [If it is worth keeping this method at all.]
9. Delete\_signup and delete\_signup\_as\_instructor have much in common and violate the DRY principle. Refactor.
10. Also see [Code Climate](https://codeclimate.com/github/expertiza/expertiza/app/controllers/response_controller.rb) issues. To reduce the size of the file, you could move some methods to sign\_up\_sheet\_helper.rb [We are trying to fix how you access Code Climate, but haven’t fixed it yet.]

**Previous version of project**

* <https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Spring_2022_-_E2223._Refactor_sign_up_sheet_controller.rb>
* <https://github.com/palvitgarg99/expertiza>
* <https://github.com/expertiza/expertiza/pull/2345>

**Comments on previous version**

The team DRYed out code used to check for certain errors; however, that was done using an array of messages and then selecting one of them using a constant subscript, whereas it would have been better to pass the messages as parameters. The compute\_signup\_topics method returns results by side-effects (in instance variables); probably this could have been done more cleanly. Most of the changes involve regularizing use of "sign\_up" vs. "signup"; this should ideally be done in a separate commit so it would be mergeable without merging the whole project. My overall judgment is that some improvements have been made, but fewer than in most other refactoring projects.

Despite the relatively low score, we can consider merging this one. It didn't do anything bad; it was just that it didn't improve as much as I would have liked to see. If we commit it, we finally fix sign\_up vs. signup, which has been a source of confusion for years. So, yes, merge if tests pass.

## E2410. View for results of bidding

**Mentor:** [Anvitha Reddy Gutha](mailto:agutha@ncsu.edu) ([agutha@ncsu.edu](mailto:agutha@ncsu.edu))

**What is needed:** When topics are opened up for bidding, students can see how “hot” each topic is by the color it has on their topic list. However, instructors have no way to view the bidding process except by impersonating students. Furthermore, when the bidding assignment algorithm is run, there is no way to verify that it did in fact assign teams to topics they had chosen.

This project is to create a field for instructors on the topic list that shows how many teams have bid for each project. Depending on how you implement it, you might or might not set up the topics to change color. Since topic lines in the topic list potentially have several fields, including partner advertisements, take care that (1) the #-of-bids field is not displayed except for projects that use bidding for topics, and (2) the field is as narrow as feasible.

| Topic | Total # of bids | # of #1 bids | # of #2 bids | Team bidding #1 |
| --- | --- | --- | --- | --- |
| Track the time | 15 | 2 | 3 | 2 team names |
| Grading audit | 5 | 1 | 1 | 1 team name |
|  |  |  |  |  |

There should be a way to see how many teams have made each topic their #1 bid, #2 bid, etc. (see above). And for any team, there should be a way to show all of its bids. There should still be a way to see the bids *after* the bidding-assignment algorithm has been run, so that it can be checked to see if it did a reasonable job of assigning teams to topics.

Here is the [original design doc](https://docs.google.com/document/d/110t5tRLUTyeqjk_RzqpsDLFRwiTN4Vfl-E8xSEIV_C4/edit?usp=drivesdk) on bidding.

**Previous implementation**

<https://expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2022_-_E2245:_View_for_results_of_bidding>

https://github.com/expertiza/expertiza/pull/2392 <https://github.com/dnguyenv/final-e2245/projects/1>

ht[tps://www.youtube.com/watch?v=OTSRqRihahQ](https://www.youtube.com/watch?v=OTSRqRihahQ)

<https://drive.google.com/drive/u/0/folders/0ALFi2P49eI5VUk9PVA?ddrp=1>

**Our comments on the implementation:** A modest amount of code is written (95 lines), and the desired functionality is implemented very well. All the code is in very appropriate classes, and SRP violations are minimized. Unfortunately there are two places where bidding needs to be checked for in the view and appropriate text rendered if it is in use. In both cases, the then clause consists of only two lines, so it probably doesn't make sense to shunt it to a different partial. While the view does make active-record calls, it does so IN THE MIDDLE of a table line, so moving the calls out of the view wouldn't be easy.

The biggest deficiency is the shallow nature of the automated tests: they just check for the case where no bids have been entered. There are no tests that bids are listed properly.

## E2411. Fix "Back" link on “New Late Policy” page

**Mentor:** [Anvitha Reddy Gutha](mailto:agutha@ncsu.edu) ([agutha@ncsu.edu](mailto:agutha@ncsu.edu))

**Background:** An instructor can create late policies for any assignment, where the instructor can specify the points per unit and the maximum penalty that can be applied for any assignment submission.

One way to reach late policy creation link is as follows:

1. Log into Expertiza as an Instructor.
2. Edit an assignment by clicking on edit logo under the “Action” column.
3. Under the “Due Date” tab click on the "New late policy" link
4. In “New late policy” fill in the required details.
5. Click “Create” to save the policy, and go to the page which shows all late policies.
6. Clicking “Back” gives an error, it should instead go to the “Due Date” tab of the assignment which was being edited.

**Issue 1:**

It is showing an error that “Failed to save the assignment: #”

Determine why this error message shows up, and fix this error so that it doesn’t occur when clicking on “New Late Policy” in the “due date” tab.

Add a test case that verifies the error doesn’t occur.

**Issue 2:**

If an instructor, while creating a policy, clicks on "Back" link and wants to go back to the previous page, (s)he sees an error instead of the “Due Date” tab of the assignment edit page (Step 2 above).

**Fix:**

The "Back" link on the "New late policy" page should direct the user to the "Due dates" tab so that they can continue to edit the assignment.

Add a test case which verifies this behavior..

**Issue 3**

On the “New Late Policy” page, once you click “Create”, it shows you a list of late policies.

The “back” link on this page should go to the “Due Date” page of the assignment which was being edited when clicked. Currently it gives error as it can’t find an assignment number.

Add a test case to check the above issues and verify it passes.

**If you are able to fix the above issues, then work on the following issues:**

On the other hand, if a user directly visits the “New Late Policy” page, without going through the “edit assignment” workflow, there isn’t a “assignment” which they should be editing. In this case, the “Back” link on both “new late policy” and “all late policies” page should go to the Assignments tab on the home screen.

Add tests to verify the above behaviors.

Previous PR reference: <https://github.com/expertiza/expertiza/pull/2482>

This was a good implementation, and the current project should continue using this as the base. Below are the comments on the previous project.

* The code lacked code comments, which could have improved the readability. Some of the code was removed, and code comments over what was removed could have been mentioned. Refactors like aid to assignment\_id help readability.

## E2412. Testing for hamer.rb

**Mentor:** Mustafa Olmez (molmez@ncsu.edu)

**Description:** hamer.rb was the file that implemented one of the “reputation systems” that can be used to determine the reliability of peer reviewers. The algorithm is described in [this paper](https://crpit.scem.westernsydney.edu.au/confpapers/CRPITV42Hamer.pdf).

This file is no longer current, having been replaced by a web service in 2015. The web-service implementation in Expertiza is described in [this paper](https://scholar.google.com/scholar?q=luggable+reputation+systems+for+peer+review%3A+a+web-service+approach&btnG=h&l=en&as_sdt=0%2C34). My [slides](https://research.csc.ncsu.edu/efg/expertiza/presentations/Research%20interests%202018.ppt) starting at Slide 15 explain the Lauw algorithm.

Refer to general testing guidelines above, but start by using the spec/lib/hamer\_spec.rb file and test that has already been written as a jumping-off point..

This project should build on the work done by [E2168](https://docs.google.com/document/d/1slx4HPIbgTH-psIKMSCF-HDF9brxf-FuYhzVT9ZiIrM/edit#heading=h.cejglmx7vhqx) and [E2212](#_zge6t02p9kis), which submitted these files.

* E2168:
* <https://github.com/expertiza/expertiza/pull/2128>
* <https://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2021_-_E2168._Testing_-_Reputations>
* <https://github.com/HenryChen34/expertiza/tree/beta>
* <https://youtu.be/sL0xQTol61w>
* E2212:
* https://wiki.expertiza.ncsu.edu/index.php/CSC/ECE\_517\_Spring\_2022\_-\_E2212:\_Testing\_for\_hamer.rb
* https://github.com/joshlin5/expertiza/projects/2
* https://youtu.be/VyeGGpxymXk
* https://github.com/expertiza/expertiza/pull/2357
* hamerAlgorithm\_\_1\_.ipynb 3826 file 2022-03-21 22:34:53 -0400

It needs to be tested with scores assigned to made-up reviews, to make sure that it is returning the “reputations” that it is supposed to return. It should be tested on cases where each reviewer submits the same scores, and on cases where reviewers submit widely disparate scores. Project E2168 was a good start, but was unable to test the values returned by the web service, because the web service was not running at that time. E2212 successfully accessed the web service; however, the returned values differed from the expected ones. As a result, they opted to mock the algorithm. It is advisable to examine the returned values.

* Expertiza : <https://docs.google.com/document/d/1hfNDT8hPmGf90FijsnWz6cx0SYUyraRxHfgrT8UVbR8/edit#heading=h.zett47k66eg2>
* The team will make setup done and read the papers to get familiar with the algorithm
  + The team will also confirm the web service is online and working
* <https://github.com/peerlogic/reputation_web_service>

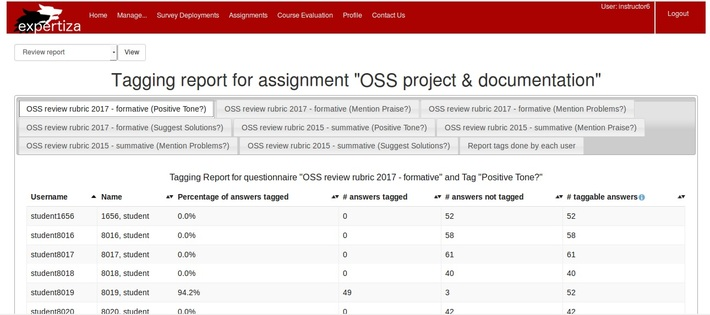
## E2413. Testing - Answer Tagging

**Mentor:** Mustafa Olmez (molmez@ncsu.edu)

Answer tagging (see [answer\_tab.rb](https://github.com/expertiza/expertiza/blob/25713567e5e5e1c0b363409715e6d8797b45b01a/app/models/answer_tag.rb#L1) and [wiki](https://expertiza.csc.ncsu.edu/index.php/Answer_tags)) helps determine a few metrics on a student’s responses to a questionnaire. Students can tag these metrics for other students for things such as “Positive Tone” or “Suggest Solutions”.

These specs below all need to be further developed to cover missing methods/lines. Some structures currently exist at the top of spec files (such as assignments, participants, etc) to help make writing tests easier. Please feel free to add to these and write extra tests to cover any edge cases you may think of. Add detailed explanations for tests.

As an instructor on an assignment, going to Etc/View Reports/Answer Tagging Report will show student answer tagging. You will be able to see the percentage of answers tagged, # of answers tagged, # not tagged, and # of taggable answers for each student on a questionnaire.



* Answer\_tags\_controller\_spec <https://coveralls.io/builds/43656335/source?filename=app%2Fcontrollers%2Fanswer_tags_controller.rb>. 0% Covered
* Tag\_prompt\_deployment\_spec <https://coveralls.io/builds/43656335/source?filename=app%2Fmodels%2Ftag_prompt_deployment.rb> 18% Covered
* New Wiki Page : <https://wiki.expertiza.ncsu.edu/index.php?title=Answer_tags>
  + Previous work for ref: <https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2021_-_E2169._Testing_-_Answer_Tagging>
  + <https://github.com/akk5597/expertiza>
  + [https://github.com/expertiza/expertiza/pull/2146](https://github.com/akk5597/expertiza)
  + [https://youtu.be/WYSUmL6jcHc](https://github.com/akk5597/expertiza)
  + [https://youtu.be/Xc7SpMhwUyc](https://github.com/akk5597/expertiza)
* Expertiza setup: <https://docs.google.com/document/d/1hfNDT8hPmGf90FijsnWz6cx0SYUyraRxHfgrT8UVbR8/edit>
* Coverage visulization:
  + Yum install lynx
  + lynx ./coverage/index.html

## E2414. Grading audit trail

**Mentor:**  Ed Gehringer ([efg@ncsu.edu](mailto:efg@ncsu.edu))

**Problem:** Any instructor can assign / edit a grade freely. There is no way of tracking who did it.  
**Web pages:**

1. Review grade: *Log in as instructor -> Manage -> Assignments -> View Review Report*

2. Submission grade: *Log in as instructor -> Manage -> Assignments -> View submissions*  
  
**What needs to be done:** A grading audit trail must be created and the following information needs to be stored:

1. When a grade is assigned by an instructor, there needs to be an indication of who did it and when it was done.
2. Comments previously provided by other instructors must also be preserved.

This information needs to be stored every time an instructor edits a grade / comment and clicks the save button.

The grading audit trail can probably pattern off of the submission records history (shown below) on Expertiza.

*The below page can be reached by logging in as instructor -> Manage -> Assignments -> View Submissions -> History*  


At the minimum, a grading log entry must include the *instructor id, assignment id, student id, grade, comment* and *timestamp*.

**Previous version of project:**

* <https://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2023_-_E2383._Grading_Audit_Trail>
* <https://github.com/expertiza/expertiza/pull/2696>
* <https://github.com/ErikLG360/expertiza>
* <http://152.7.179.44:8080/>
* <https://drive.google.com/file/d/1HIrJtCsbZXSqGPHShuV6mbSTOKvV5MxT/view?usp=drive_link>
* https://expertiza.csc.ncsu.edu/index.php/CSC/ECE\_517\_Spring\_2023\_-\_E2329.\_Grading\_audit\_trail(Project\_4)
* http://152.7.177.175:8080
* https://github.com/chetana-c/expertiza
* <https://github.com/expertiza/expertiza/pull/2557>
* https://youtu.be/CMVBVctHcqE

**Our comments on the implementation:** Lacks comments and minor changes required for the method names. The grading history page doesn’t show the assignment name and the group/user that it is correctly displaying. It needs some additional changes for it to be in a working state.

The comments on the demo were addressed and fixed today.

I think the tests are not enough and no new tests were added to fix it.

## E2415. Reimplement response[s]\_controller.rb

**Mentor:** [Ameya Vaichalkar](mailto:agvaicha@ncsu.edu)

The naming convention for controllers dictates that they be named in the plural, but Expertiza predates that convention and uses the singular for a few controllers. The reimplementation is to follow convention.

The current ResponseController is very long, and has many functions besides the basic CRUD methods.

The authentication and authorization methods are up to date, and should be kept without change.

The edit method contains functionality to sort reviews, which would be better placed in the Response model. Also, it checks whether the reviewer is a team; it would be better to implement this functionality elsewhere, preferably by polymorphism, so it would not have to be checked via an if statement here and elsewhere.

It uses multiple methods to set parameters (assign\_action\_parameters, set\_content). Although these methods are much better than duplicating code elsewhere, there may be a more elegant way to assure that client methods have access to the objects that they need.

new\_feedback is used to create author feedback to send to a reviewer about a review. It is good for it to be in this controller, since it also creates a Response object.

Items (Questions) always need to be displayed in order. In the controller and helper, there are 4 calls to sort\_questions, whose implementation is only a single line. Since items only need to be sorted right as they are being displayed, it is not clear why four calls are needed. BTW, the standard term for components of a rubric is “item”, not “question”, since many of them are not framed as questions. Please use the term “item” in your code.

The method questionnaire\_from\_response\_map decides what kind of rubric (or quiz or survey) is in use. There is a possibility that the review rubric may [depend on what topic is being reviewed](https://expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Fall_2021_-_E2147._Role-based_reviewing), and the method needs to handle this case. Nonetheless, it may be possible to simplify it.

The controller also contains a method to generate an email from a reviewer to an author. This is but one of several kinds of emails that are sent pursuant to responses. Other emails are defined in response.rb. Emails should be generated in a helper, which can be the same helper that response.rb uses. Emails should be sent …

* to the team of authors when a reviewer submits a review of the authors’ work,
* to the reviewers when an author submits a new version of their submission,
* to the instructor when two review scores differ by more than a “threshold” amount, and
* from the reviewer to the author whenever the reviewer wants (authors should also be allowed to respond in a double-blind fashion to reviewers).

You should recode the existing methods in a helper, but you need not implement email methods that are currently missing from Response and ResponseController.

**What to code.** The new ResponsesController should create and edit responses. As part of creating a response, it needs to create answer blanks (Answer objects) for each of the items on the rubric (or survey or quiz). When creating the answer blanks, it needs to cycle through the Questions on the Questionnaire in the order specified by their sequence numbers. I think it will be unnecessary to write an AnswersController, because Answer objects are created only in conjunction with Responses. After all the Questions have been placed on the view, the “Additional comment” field of the Response object should be rendered.

The various Question objects may represent different kinds of rubric items. Each kind is implemented in a subclass of Question (read about the subclasses [here](https://docs.google.com/document/d/11TqWr-L5qjd2lHSUwWKmvG4W5jNHy1j2QrZPxnzc7BQ/edit#heading=h.gczmekdk7815); search for “Fields in questions table”). It is not your job to code these classes, but you do need to be careful to render the right kind of answer blank (checkbox, dropbox, text field, text area, etc.). These are defined by instance methods in the Question subclasses. You can assume that the same instance methods will be used as in the current Expertiza code.

The ResponsesController should also reimplement the methods that return information about the response, e.g., questionnaire\_from\_response, questionnaire\_from\_response\_map. If you do not understand what some method is doing, you can search the repo to see if it is being used. If it is never called (a “dead” method), then remove it. If it is called, try to figure out what it is doing, and if in doubt, ask me. Methods that are in use should, in general, be re-written in the new ResponsesController.

## **E241**6**. Reimplement the Question hierarchy**

**Mentor:** Ali Qureshi (mquresh@ncsu.edu)

The Question class and its subclasses are used to implement all rubric items, quiz questions, and survey questions in Expertiza. Here is the hierarchy.

* Choice question
  + Scored question
    - Scale [√]
    - Criterion [√]
  + Unscored question
    - Dropdown [√]
    - MultipleChoice
    - CheckBox [√] [should this be a Scored question?}
* TextResponse
  + TextArea [√]
  + TextField [√]
* UploadFile [√]

Dropdown, MultipleChoice, and Scale questions have a lot in common: you display the alternatives and let the user choose one of them. They also differ, in that only Scale is required to have a numeric sequence of choices; Dropdown and MultipleChoice may, but also may have selections that don’t correspond to anything numeric. We would like to write a single piece of code to display all three types of questions.

Previous team’s links:

https://wiki.expertiza.ncsu.edu/index.php/CSC/ECE\_517\_Fall\_2023\_-\_E2374.\_Reimplement\_the\_Question\_hierarchy

https://youtu.be/lDqATi8p0Jw

https://github.com/Rishi2812/reimplementation-back-end

https://github.com/expertiza/reimplementation-back-end/pull/59

Previous team’s comments: The team has successfully reimplemented the question hierarchy. The models for the questions were earlier returning html code which was not correct due to the new repo having a separate backend and frontend and now they return the correct json responses. Test cases have been written well and the test skeleton was made use of which is great. The pull request has a lot of files because of whitespace changes which makes it harder to merge.

## **E241**7**. Reimplement the SubmittedContentController**

Mentor: [Ameya Vaichalkar](mailto:agvaicha@ncsu.edu)

**Project Overview:**

Reimplement SubmittedContentController from the [Expertiza](https://github.com/expertiza/expertiza) to the [Reimplementation-back-end](https://github.com/expertiza/reimplementation-back-end) repository. The SubmittedContentController handles various functionalities related to managing submitted content, including file submissions, file uploads, hyperlink submissions, folder actions, and downloads.

Current Implementation: [SubmittedContentController](https://github.com/expertiza/expertiza/blob/main/app/controllers/submitted_content_controller.rb)

**What needs to be done**:

1. The goal of the reimplementation is that Participants should be able to receive a message and a suitable HTTP code upon submitting/removing hyperlinks, submitting/uploading a file for an assignment, checking the contents of the uploaded file and file type, and deleting the uploaded file.
2. There are various methods in the controller that perform more than one responsibility, identify the DRY code, and avoid it in the reimplementation.
3. Some methods use instance variables for defining every variable but only use them where they are required.
4. Create the required models, such as submission records, etc., with the attributes from the current implementation if they are not already present in the reimplementation repo.
5. Return proper **HTTP codes and descriptive and short messages** for **every** controller method.
6. Skip the authorization logic and log from the code for now.
7. Feel free to add the necessary helper methods from the file helper mixin.
8. Include comments for every method.
9. Do not modify any data attributes from the current implementation, for example, db migrations to add/modify attributes.
10. There are instances of the Law of Demeter violations where the controller directly calls methods on objects multiple levels deep. For example, accessing participant.team.path.to\_s in the submit\_file method. Refactor the code to avoid this.
11. Use [Swagger](https://docs.google.com/document/d/1aOZsZd8cweT8lFcUcJ028OcT_qmGoNYYkbifVkZV_Yk/edit) to **test all the controller methods**.

## **E241**8**.** Reimplement due\_date.rb

**Mentor:** [Kalyan Karnati](mailto:kkarnat@ncsu.edu)([kkarnat@ncsu.edu](mailto:kkarnat@ncsu.edu))

The due date model class in the Expertiza application is responsible for managing various types of due dates, such as assignment submission deadlines, peer review deadlines, or topic due dates. While the present codebase is relatively clean and does not exhibit conflated responsibilities, there is still room for improvement, particularly in terms of refactoring methods and reducing the reliance on class methods.

Here are the specific tasks that need to be addressed in the reimplementation:

* Convert class methods to instance methods where applicable. For example, the `set\_duedate` method should potentially accept a `duedate` parameter as an instance or call `set\_duedate` on a `duedate` object.
* Remove any methods and fields that are only used in test files and do not contribute to the application's functionality.
* Ensure method names describe the responsibility they accomplish rather than how they achieve their goal.
* Utilize Rails built-in APIs for tasks such as validation, scoping, and database access.
* Follow Ruby conventions for each method, adhere to the DRY principle, and ensure each method has a single responsibility.
* Consider moving some class methods to a helper class or module dedicated to handling due dates rather than defining them directly on the DueDate model.
* Evaluate whether the logic in the `done\_in\_assignment\_round` method is better suited for a service object or another class responsible for handling responses.
* Override the comparator operator for the DueDate class and replace the `deadline\_sort` class method.
* Write tests for any updated methods and ensure existing tests are updated if they fail due to changes in the refactored code.
* Utilize code quality tools like Code Climate to ensure there are no major code smells in the reimplementation.
* By addressing these tasks, we aim to improve the readability, maintainability, and efficiency of the DueDate model class within the Expertiza application.  
    
  Can refer:  
  <https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2023_-_E2368._Reimplement_of_due_date.rb>

**What needs to be done:**  
Please follow these instructions for the setup of the project:  
[Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end) and

[Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)

## E2419. Reimplement duties controller.rb and badges controller.rb

**Mentor:** [Chunduru Chetana](mailto:<cchetan2@ncsu.edu>)[cchetan2@ncsu.edu](mailto:cchetan2@ncsu.edu)

**Background:**

The project involves the reimplementation and enhancement of the Duties and Badges modules within an API framework built on Rails. The duties\_controller.rb defines actions for managing duties, including creation, editing, updating, and deletion of duties in the Expertiza system. On the other hand, the badges\_controller.rb handles the creation of Badge instances, allowing users to specify parameters for the badge and optionally attach an image file, which is then saved and associated with the badge.

The focus of the project lies in ensuring that these functionalities are implemented as RESTful API endpoints, adhering to JSON response standards. Thorough testing, including both positive and negative scenarios, is crucial to validate the correctness and robustness of the implemented functionalities. Moreover, attention to security, error handling, and validation is essential to maintain data integrity and prevent potential vulnerabilities. The reimplementation should align with Rails conventions and best practices, fostering maintainability and scalability while ensuring seamless integration with other modules of the Expertiza system.

**Important Points:**

* Reimplement the functionalities for Duties and Badges modules as API endpoints returning data in JSON format.
* Ensure thorough testing covering both pass and fail scenarios for all REST endpoints.
* Pay attention to error handling and validation to maintain data integrity.
* Follow Rails conventions and best practices for controller and model implementations.
* Consider security aspects, such as input validation and authorization, to prevent potential vulnerabilities.
* Document the API endpoints and their functionalities for easy reference.
* Implement versioning for the API endpoints if required to support future updates and changes.
* Consider scalability and performance optimizations, especially if dealing with a large volume of data.
* Collaborate with the team to ensure seamless integration with other modules and components of the application.
* Regularly review and refactor code to maintain code quality and readability.
* Provide comprehensive documentation and support for future maintenance and enhancements.
* Stay updated with the latest developments in Rails and relevant technologies to leverage new features and improvements effectively.

Can Refer:

<https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2023_-_E2369._Reimplement_duties_controller.rb_and_badges_controller.rb>

<https://github.com/expertiza/reimplementation-back-end/pull/39>

<https://wiki.expertiza.ncsu.edu/index.php?title=Debugging_Rails>

**What needs to be done:**  
Please follow these instructions for the setup of the project:

[Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)

## E2420. Reimplement student\_quizzes\_controller.rb

**Mentor:** [Chunduru Chetana](mailto:<cchetan2@ncsu.edu>)[cchetan2@ncsu.edu](mailto:cchetan2@ncsu.edu)

**Project Overview:**

The team is tasked with refactoring the student\_quizzes\_controller.rb in the reimplementation-backend repository. This controller handles various actions related to creating, scoring, and recording responses for quizzes taken by reviewers or students within the context of the same assignment. The current implementation violates key Rails design principles, including the DRY principle, and contains methods that should ideally reside in model classes. Additionally, there are issues with code repetition, unclear method comments, and ambiguous variable names that need to be addressed. The goal of this project is to refactor the controller to adhere to best practices while ensuring that existing functionalities remain unchanged.

**Important Points:**

* Refactor the student\_quizzes\_controller.rb to adhere to essential Rails design principles, including the DRY principle.
* Identify methods that should be moved to model classes for better organization and adherence to MVC architecture.
* Improve method comments to enhance code readability and clarity of functionality.
* Address code repetition by consolidating shared code into reusable functions.
* Review and potentially rename variables to better reflect their purpose and improve code comprehension.
* Ensure that existing tests pass after the refactor and create new tests as needed to cover the refactored code.
* Manual verification may be required due to the limited test coverage in the existing codebase.
* Focus on refactoring without introducing changes to existing functionalities, ensuring that core functions remain unaffected.

**Previous Wiki:**

**​​**<https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2023_-_E2376._Reimplement_student_quizzes_controller.rb>

**What needs to be done:**  
Please follow these instructions for the setup of the project:

<https://github.com/expertiza/expertiza/blob/main/app/controllers/student_quizzes_controller.rb>

## E2421. Reimplement impersonating users (within impersonate\_controller.rb)

**Mentor:** [Chunduru Chetana](mailto:<cchetan2@ncsu.edu>)[cchetan2@ncsu.edu](mailto:cchetan2@ncsu.edu)

**Project Overview:**

The objective is to reimplement the backend code for the impersonation feature in the new implementation of Expertiza. The current implementation relies on sessions, which won't be compatible with the new implementation using JWT tokens for authentication and returning JSON responses. The challenge lies in transitioning the logic from session-based management to JWT-based authentication while maintaining the functionality of impersonating users. The reimplementation involves planning how the backend will communicate with the frontend, potentially requiring changes in existing or new files beyond the impersonate\_controller.rb.

**Important Points:**

* Reimplement the impersonation feature from the current implementation of Expertiza into the new backend codebase.
* Adapt the logic from session-based management to JWT-based authentication for compatibility with the new implementation.
* Ensure that the quality of comments, particularly in the impersonate\_controller.rb, is maintained for better code understanding and maintainability.
* Remove any unnecessary code or refactor it for improved efficiency.
* Consider how the backend will interact with the frontend, potentially using the browser to store backup sessions and sending them in the request body when needed.
* Focus on implementing the functionality of impersonating users while adhering to the API-only nature of the new application and returning responses in JSON format.

**Resources:**

<https://github.com/expertiza/expertiza/blob/main/app/controllers/impersonate_controller.rb>

<https://wiki.expertiza.ncsu.edu/index.php?title=CSE/ECE_517_Spring_2021_-_E2108._Impersonate_controller.rb>

**Latest wiki page:**

<https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2023_-_E2377._Reimplement_impersonating_users_(functionality_within_impersonate_controller.rb)>

**What needs to be done:**  
Please follow these instructions for the setup of the project:

[Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)

## **E24**22**. Reimplement questionnaire.rb**

Mentor: [Anuj Naresh Chetwani](mailto:achetwa@ncsu.edu)

**Project Overview:**

The project includes re-implementation of the questionnaire.rb found in [Expertiza](https://github.com/expertiza/expertiza/blob/main/app/models/questionnaire.rb) to the [Reimplementation-back-end](https://github.com/expertiza/reimplementation-back-end) . The current implementation includes various functionalities such as validation checks and methods. The goal is to implement these ensuring the code follows SOLID principles and is DRY in nature.

Important points

1. The new model should use rails in built active record validation and eliminate the

Validate\_questionnaire method.

1. The model should check for any overlapping functionalities from the controller and ensure all business logic is present in the model only.
2. Eliminate methods and fields from this model that are exclusively utilized in test files so that only the required changes to the model are merged into the main branch.
3. Ruby naming conventions should be followed for each method.
4. Tests should be performed using Rspec covering all necessary functionalities.
5. Comments should be present for every new functionality that is added and ensure to avoid all mistakes mentioned in the checklist document.
6. Simplification of code is possible in various functions such as get\_weighted\_score and compute\_weighted\_score. For more details check the [PR](https://github.com/expertiza/reimplementation-back-end/pull/27) which has changes and ideas used for the implementation by the last team working on it.
7. There is already an existing model in the [Reimplementation back end](https://github.com/expertiza/reimplementation-back-end/blob/main/app/models/questionnaire.rb) . However this is essentially a copy paste from expertiza. This was done by the team working on the controller last spring in 2023. All changes should be made to the same file. Starting from scratch in your fork will be the quickest way to ensure no mix up from the existing implementation.

## **E24**23**. Reimplement the response\_map hierarchy**

Mentor: [Anuj Naresh Chetwani](mailto:achetwa@ncsu.edu)

**Project Overview:**

When someone writes a review in Expertiza, they are creating an instance of the Response class. Each Response is created based on a particular [ResponseMap](https://wiki-expertiza.csc.ncsu.edu/index.php/Response_maps). The ResponseMap tells who the reviewer is (the reviewer\_id), who the reviewee is (the reviewee\_id), and what is being reviewed (the reviewed\_object\_id).

* The reviewer\_id is normally a record in the Participants table, which specifies an AssignmentParticipant. However, if teams rather than individuals are doing reviews for this assignment, it may be an AssignmentTeam.
* For reviews of student work, the reviewee\_id is always an AssignmentTeam. (If an assignment is set up so *individuals* rather than teams submit work, the reviewee will still be a 1-member AssignmentTeam.)
* For teammate reviews, the reviewee\_id is always an AssignmentParticipant.
* The reviewed\_object\_id is normally an Assignment. But in the case of *meta-reviews*, the reviewed\_object\_id is another ResponseMap (because it is a review that is being reviewed).

There are about 8 different types of ResponseMaps in the current Expertiza, but for this project, you will only re-implement four of them, ReviewResponseMaps, FeedbackResponseMaps, and TeammateReviewResponseMaps and an additional one after discussion it with your mentor. The first three i.e ReviewResponseMaps, FeedbackResponseMaps, TeammateReviewResponseMaps have already been worked upon and the relevant information for the changes done can be found in the following [PR](https://github.com/expertiza/reimplementation-back-end/pull/21) .

Some more information about the project:

The ResponseMap hierarchy is plagued by a surfeit of class methods. Two of those in the superclass, assessments\_for and reviewer\_assessments\_for, return Responses, and seem better suited to reside in response.rb. The other one, comparator, should be replaced by an overloading of the <=> operator. In ReviewResponseMap, the export operation should be reimplemented based on [this framework](https://wiki-expertiza.csc.ncsu.edu/index.php/CSC517_Spring_2019/E1923_New_Framework_For_Import/Export). It will still be a class method, but should be much simpler.

In ReviewResponseMap, the get\_responses\_for\_team\_round method belongs in the Response class. So do several of the other methods that return Reponses. My suggestion is to create a partial implementation of the response.rb that includes only these methods. It will be easy to merge with the response.rb being reimplemented by E2312. In general, most methods whose name begins with get should be renamed to drop the “get”, in accordance with Ruby convention.

TeammateReviewResponseMap and FeedbackResponseMap exhibit fewer design problems, but some of their methods can be renamed according to Ruby convention too. BTW, FeedbackResponseMap is used for giving author feedback—feedback from the author to the reviewer about the quality or helpfulness of the review.

Important points:

1. All existing associate code that needs to be implemented for different maps can be found in [expertiza](https://github.com/expertiza/expertiza/tree/main/app/models).
2. The test suite should cover all functionalities and be well commented ensuring that it is easy to understand which test corresponds to what scenario.
3. Be sure to eliminate methods and fields that are exclusively utilized in test files.
4. The changes will be evaluated and checked for SRP which should be followed to ensure that the code can be merged.
5. The existing [PR](https://github.com/expertiza/reimplementation-back-end/pull/21) is a reference and the [wiki](https://wiki-expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2023_-_E2338._Reimplement_the_response_map_hierarchy) page explains the design well, however, the implementation can be improved to follow SRP, eliminating class methods and removing unnecessary files.

## E2424. Reimplement the Bookmarks Controller

**Mentor:** Ali Qureshi (mquresh@ncsu.edu)

**Background:** The BookmarksController on Expertiza serves as the backend logic for bookmark management and user interactions. It features actions for listing bookmarks associated with a specific topic, creating new bookmarks, editing existing ones, and deleting bookmarks. Authorization rules, defined in the action\_allowed? method, ensure that users have appropriate roles and permissions for each action. The controller includes functionality for users to get bookmark ratings , with methods like bookmark\_rating and save\_bookmark\_rating\_score. Additionally, it calculates and displays average scores for specific and total ratings using the specific\_average\_score and total\_average\_score methods, respectively. Strong parameters are handled through private methods (create\_bookmark\_params and update\_bookmark\_params). The controller logs various actions and provides feedback to users through flash messages. Overall, the BookmarksController orchestrates the interaction between users, bookmarks, and ratings within the application.

What needs to be done?

* The controller should have the CRUD operations to create and edit bookmarks based on whether the user has adequate permission. The bookmarks controller should have methods for creating, updating, deleting, and viewing bookmarks.
* The code should be written in a clean and concise manner. Methods with identical names that perform different functionalities should be renamed for clarity. Functions or functionality that are not clear should be commented on or removed. Any loops or methods that can be refactored for better performance should be addressed e.g. delete all questions that belong to a questionnaire
* Any unused or unclear functionality should be removed from the controllers. This will help to reduce complexity and make the code easier to maintain.

## Tests should be written for the bookmarks controller. The tests should cover at least 80% of the code, and tools like Rubocop and Code Climate should be used to verify code smells.

## E2425. Create a Courses user interface in ReactJS

Mentor: Riya Gori (rvgori@ncsu.edu)

Can refer:

<https://wiki.expertiza.ncsu.edu/index.php?title=CSC/ECE_517_Fall_2023_-_E2385._Create_a_Courses_User_interface_in_ReactJS>

https://github.com/shreyavaidya2311/reimplementation-front-end

https://github.com/expertiza/reimplementation-front-end/pull/26

https://www.youtube.com/watch?v=juo7aEv9Ax0

The current Courses User Interface has been identified to incorporate an inappropriate set of widgets leading to suboptimal user experience and performance issues. The reimplementation will focus on:

* **Intuitive Course Management**: Develop a streamlined interface for easy navigation and management of courses.
* **Enhanced Widgets**: Incorporate optimized widgets ensuring functionality and user engagement.
* **Responsive Design**: Ensure the interface is accessible across various devices enhancing user experience.
* **Performance Optimization**: Enhance loading speeds and overall performance metrics.

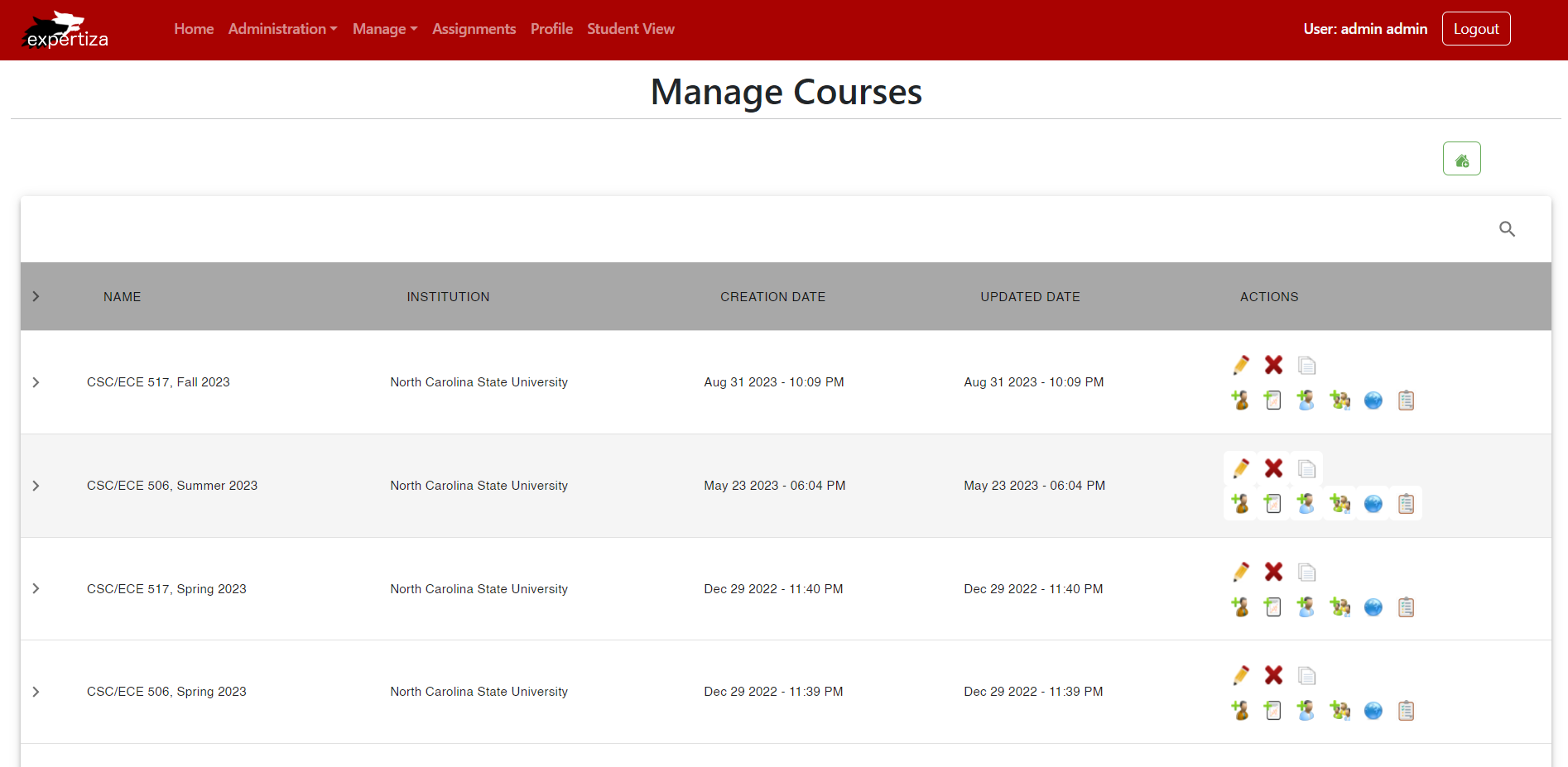
Prerequisite: React.js

Project Setup Instructions:

Issues:

The widgets are not homogenous with the other widgets throughout the application.

The issue can be resolved by referring to the frontend of the new implementation.



**What needs to be done:**  
Please follow these instructions for the setup of the project:  
[Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end) and

[Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)

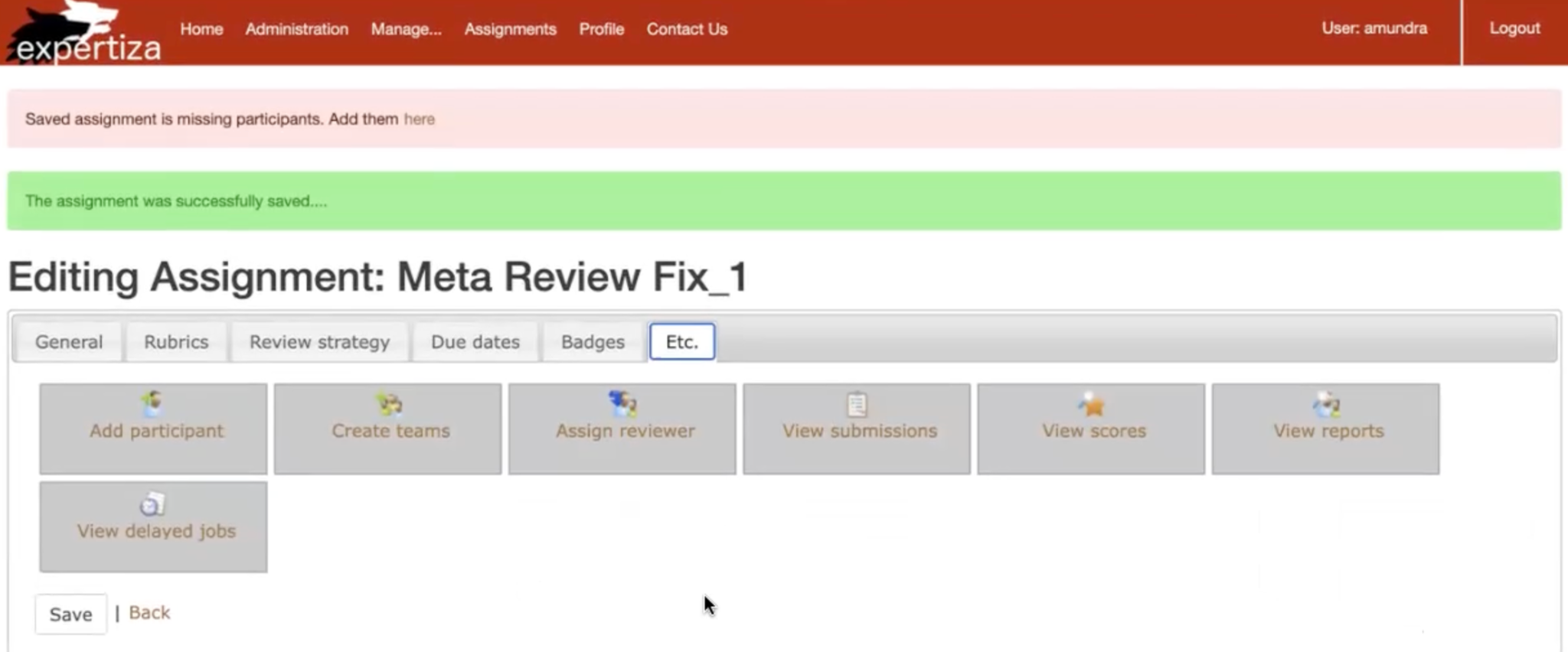
## E2426. Create a UI for Assignment Edit page "Etc" tab in ReactJS

**Mentor:** [Kalyan Karnati](mailto:kkarnat@ncsu.edu)**(** [kkarnat@ncsu.edu](mailto:kkarnat@ncsu.edu))

Create a Etc page which displays all the miscellaneous tasks for the Editing Assignment option.

Path: Go to Manage -> Assignment -> Edit (pencil icon) -> Etc

This UI is good and can be reconstructed in a similar way but feel free to exercise your creativity while making this page easy to understand and navigate.



Can refer <https://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_517_Fall_2023_-_E2363._Create_a_UI_for_Assignment_Edit_page_%22Etc%22_tab_in_ReactJS>

For this particular project you need to have a basic understanding of React and Typescript.

**What needs to be done:**  
Please follow these instructions for the setup of the project:  
[Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end) and

[Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)

## E2427.UI for questionnaire.rb

Mentor: Riya Gori (rvgori@ncsu.edu)

**Problem:** The current non-typescript front end for the questionnaire UI needs to be refactored to use Typescript.

**Previous links:**

<https://expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2023_-_E2343._Questionnaire_UI>

https://github.com/varundeepakgudhe/reimplementation-front-end

https://github.com/expertiza/reimplementation-front-end/pull/6

https://youtu.be/DuJj8LSlhwQ

**Steps:**

Refactor the existing code: The current codebase is written in non-typescript front end. The first step would be to refactor this code to use Typescript. This would involve converting all the JavaScript files to TypeScript and ensuring that the code is still functional after the conversion.

Implement Typescript in the UI: The UI for the questionnaire model has been implemented in React. You would need to ensure that this UI is compatible with Typescript. This might involve rewriting some parts of the UI code in Typescript.

Ensure functionality remains intact: The create, update, delete functionalities are currently working well and appropriate validations have been put in place for the required fields. After the refactoring, you need to ensure that these functionalities still work as expected.

Update the JSON server: The team has mocked a JSON server which is currently dealing with all the data. After the refactoring, you need to ensure that the JSON server can still handle all the data correctly.

Testing: After all the changes, comprehensive testing needs to be done to ensure that all parts of the application are working as expected. This includes unit tests, integration tests, and end-to-end tests.

Documentation: Finally, update the project documentation to reflect the changes made during the refactoring. This includes updating the code comments, the README file, and any other relevant documentation.

**Prerequisites: React.js, Typescript**

## E2428.Replicate Roles and Institution UIs in ReactJS

Mentor: Riya Gori (rvgori@ncsu.edu)

Problem: The current project is implemented in Ruby on Rails and needs to be re-implemented using TypeScript and ReactJS.

Web pages:

1. Roles UI: Log in -> Manage -> Roles
2. Institution UIs: Log in -> Manage -> Institutions

What needs to be done: The Roles and Institution UIs need to be replicated in ReactJS. The new implementation should have the same functionality as the current one but should be written in TypeScript and use ReactJS.  
Understand the Existing System: The first step is to understand the existing Roles and Institution UIs in the current Ruby on Rails application. This involves understanding the data models, the controllers, and the views associated with these UIs.

1. Setup the Development Environment: Set up a development environment with TypeScript and ReactJS. This will be where you will build the new UIs.
2. Design the Components: Based on your understanding of the existing system, design React components that will make up the Roles and Institution UIs. Each component should correspond to a logical part of the UI and should manage its own state and props.
3. Implement the Components: Write TypeScript code to implement these components. The code should replicate the functionality of the existing UIs. This includes all the features present in the current implementation.
4. Test the Components: Write tests for your components to ensure they work as expected. This can involve unit tests, integration tests, and end-to-end tests.
5. Integrate the Components: Once the components are implemented and tested, they need to be integrated into the larger application. This involves setting up routes for the new UIs and ensuring they work with the existing backend.
6. Review and Refactor: Review your code for any potential improvements. Refactor where necessary for better readability, maintainability, or performance.

At the minimum, the reimplementation must include all the features present in the current implementation.

Previous version of project:

https://expertiza.csc.ncsu.edu/index.php/CSC/ECE\_517\_Spring\_2023\_-\_E2348\_Replicate\_Roles\_and\_Institution\_UIs\_ReactJS

https://github.com/palash27/reimplementation-front-end/tree/Team-3464

https://github.com/expertiza/reimplementation-front-end/pull/7

https://www.youtube.com/watch?v=nkwzycUgwqw

## E2429. Reimplement student\_task list

**Mentor:** Kashika Malick (kmalick@ncsu.edu)

**Project Overview:**

The Expertiza application is a collaborative platform used by students and instructors for managing assignments, peer reviews, and feedback. The current student task list interface lacks responsiveness, usability, and performance. The objective is to reimplement the frontend of the student task list using React JS and TypeScript. This reimplementation aims to enhance the user experience, improve task management, and optimize performance.  
  
The project will focus on the following features:

1. Dynamic Task Table: Display a table listing student assignments.  
   Columns: Assignment name, course, topic, current stage, review grade, badges, stage deadline, and publishing rights.  
   Implement sorting and filtering functionalities for efficient task navigation.
2. Responsive Design: Prioritize accessibility and user-friendly layouts.
3. Lazy Loading: Optimize performance by loading content only when necessary. Improve page load times for a smoother user experience.

**Prerequisite**: For this particular project you need to have a basic understanding of React and Typescript.

**Project Setup Instructions:**  
Please follow these instructions for the setup of the project:

* [Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end)
* [Reimplementation - Backen](https://github.com/expertiza/reimplementation-back-end)

## 

## E2430. Reimplement student\_task view

**Mentor:** Kashika Malick (kmalick@ncsu.edu)

**Project Overview:**

The student task view within the Expertiza application provides students with essential information about their assignments, deadlines, and progress. However, the current interface lacks responsiveness and user-friendly features. Our objective is to reimplement the frontend of the student task view using React JS and TypeScript. This reimplementation aims to enhance usability, provide real-time updates, and improve overall student experience.  
  
The project will focus on the following features:

1. Task Details Display: Show detailed information about a specific task, including assignment name, due date, current stage, review status, and any associated badges.
2. Task Interaction: Implement interactive elements such as buttons for submitting assignments, requesting revisions, and viewing feedback. Allow students to view the different stages of the task.
3. Timeline: Show a visual representation of the timeline containing all the due dates and also a dynamic navigation visualizing present day and time.
4. Lazy Loading: Optimize performance by loading content only when necessary. Improve page load times for a smoother user experience.

**Prerequisite**: For this particular project you need to have a basic understanding of React and Typescript.

**Project Setup Instructions:**  
Please follow these instructions for the setup of the project:

* [Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end)
* [Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)

## 

## E2431. Reimplement grades/view\_team

**Mentor:** Kashika Malick (kmalick@ncsu.edu)

**Project Overview:**

The primary goal of this project is to reimplement the front end for the grades/view\_team page in Expertiza. The existing page lacks optimal performance and modern design aesthetics, leading to decreased usability. Our objective is to enhance the user experience and interface efficiency by creating a revamped front end using React JS and TypeScript.  
  
The project will focus on the following features:

1. Team Information Display: Show information about team names created for an assignment and relevant assignment details
2. Reviewers and Feedback: List all reviewers associated with each team. Display feedback provided by reviewers for each assignment. Ensure consistency between the list of reviewers and the feedback section.
3. Visual Representation of Scores: Create visual charts or graphs to represent review scores. Compare scores across different teams and assignments.
4. Assignment Status Indicators: Indicate the current status of each assignment (e.g., draft, in progress, archived). Provide color-coded labels or icons for easy identification.
5. Dynamic Filters and Sorting: Allow instructors to filter assignments based on criteria such as team creation, submission, and review deadlines. Implement sorting options for efficient data exploration.

**Prerequisite**: For this particular project you need to have a basic understanding of React and Typescript.

**Project Setup Instructions:**  
Please follow these instructions for the setup of the project:

* [Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end)
* [Reimplementation - Backen](https://github.com/expertiza/reimplementation-back-end)

## E2432: UI for Participants.rb

**Mentor:**  [Kalyan Karnati](mailto:kkarnat@ncsu.edu)([kkarnat@ncsu.edu](mailto:kkarnat@ncsu.edu))

**Project Overview:** Managing participant information within web applications can be a cumbersome task, often requiring users to navigate through multiple pages or menus to perform actions such as creating, updating, or deleting participant records.

**Objectives:**

* Develop Typescript modules for managing participant information, including creation, updating, and deletion functionalities.
* Design a centralized main page for accessing all participant management functionalities.
* Implement a mock JSON server to simulate a database environment for storing participant data.
* Ensure user-friendliness and efficiency in participant management operations.

**Expected Outcome:**

The proposed participant management system will streamline the process of managing participant information within web applications, enhancing user experience and productivity. By providing intuitive access to creation, updating, and deletion functionalities from a centralized main page, users will be able to efficiently manage participant records without unnecessary navigation. Additionally, the mock JSON server will enable seamless interaction with participant data, ensuring a smooth user experience.

**Can refer:** <https://wiki-expertiza.csc.ncsu.edu/index.php/CSC/ECE_517_Spring_2023_-_E2341._UI_for_Participants_Model>

For this particular project you need to have a basic understanding of React and Typescript.

**What needs to be done:**  
Please follow these instructions for the setup of the project:  
[Reimplementation - Front End](https://github.com/expertiza/reimplementation-front-end) and

[Reimplementation - Backend](https://github.com/expertiza/reimplementation-back-end)