# Continuing the Development of E-TRIALS

By

**Tim McCarthy**

18th of March, 2021

Presented to:
Dr. Neil Heffernan
Dr. Korinn Ostrow
Mr. Ryan Emberling

An Interactive Qualifying Project submitted to the Faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the degree of Bachelor of Science

# Abstract

In the midst of the COVID-19 pandemic, online learning has become an essential means of teaching students. In light of this, the importance of identifying how to best support students learning remotely has never been more critical. One tool that allows researchers to accomplish this is E-TRIALS (the EdTech Research Infrastructure to Advance Learning Science). Built on top of the online learning system ASSISTments, E-TRIALS is an interactive web application that allows researchers to run randomized controlled trials to contrast different learning interventions. During a preceding IQP, the E-TRIALS team constructed a design for the E-TRIALS application and began its development, supported by grant funding from Schmidt Futures. In this IQP report, I describe the continued development process over the last 6 months and how the IQP and my role have influenced development, the current status of E-TRIALS, and future directions. The application has come a long way since the prior report, and although it is not finished yet, a minimum viable product is slated to be released in April 2021.

# Table of Contents

# Introduction

## The ASSISTments Platform

ASSISTments is an online learning system that helps teachers provide assistance to and receive assessments of their students. Teachers can build their own educational content or draw from certified textbook or Open Educational Resource (OER) materials, and assign those materials to their students through an integrated learning management system, such as Google Classroom or Canvas. After students complete their assignments, teachers receive student and class reports that help them gauge their students' ability and adjust their class instruction accordingly (Ostrow, 2015). ASSISTments also provides a variety of student supports including immediate feedback after completing problems, scaffolding (which breaks down a problem into multiple substeps), hints (which progressively provide more information toward the correct solution), and common wrong answer messages (which offer support tailored to specific wrong answers). These student supports strengthen students' ability to understand their mistakes quickly and encourage them to keep trying to solve the problem. Currently, ASSISTments is being used by more than 20,000 teachers to teach K-12 core math curricula to more than 500,000 students around the world (Ostrow & Emberling, 2020).

The ASSISTments platform is also unique in that it provides an underlying structure for performing randomized controlled trials (RCTs). With this structure, researchers can perform comparative AB testing at scale without extensive programming experience or knowledge of the complexities of the ASSISTments architecture. However, the existing research structures that comprise ASSISTments are still very complex and RCTs can be difficult to assemble. Not only do researchers have to put in many hours to completely build a study, but in times of confusion, the ASSISTments staff also have to lend their time to support researchers' efforts. In order to make learning science research more cost effective and scalable, members of the ASSISTments project at WPI, together with The ASSISTments Foundation, are developing the E-TRIALS application to streamline the process of experimental design and development. My IQP project, as detailed herein, was in support of these efforts.

## E-TRIALS

The ASSISTments platform provides for the ability to conduct RCTs through the E-TRIALS (EdTech Research Infrastructure to Advance Learning Science) project. The new E-TRIALS application is a learning research platform built to leverage many existing features of the ASSISTments RCT structure. It allows researchers to design and deploy randomized controlled trials. E-TRIALS provides researchers access to thousands of test subjects (student users of ASSISTments) while remaining free or extremely inexpensive. Furthermore, E-TRIALS maintains its own experimental structures, so researchers can worry less about how the study will be built and focus more on the content of their experiment.

E-TRIALS can be broken down into two main functions: study creation and data retrieval. When launched, the application will allow researchers to easily create studies to assess methods of learning according to specially designed templates that work with ASSISTments. E-TRIALS then generates, aggregates, and organizes data in order to help researchers understand the results of the assessments.

# The Development of E-TRIALS

## The 1.0 Builder

Before development on the E-TRIALS application began, studies could be created in the ASSISTments builder (hereafter referred to as the 1.0 builder), a barebones tool that researchers used to launch their RCTs on the ASSISTments platform. While the 1.0 builder got the job done, researchers had a difficult time developing their studies because the tool was meant to allow teachers to build simple problem sets while also allowing researchers to utilize advanced structures to build RCTs. This meant the builder had to incorporate features associated with both content building and study building, which led to a user interface that was difficult to use, as shown in Figure 1.

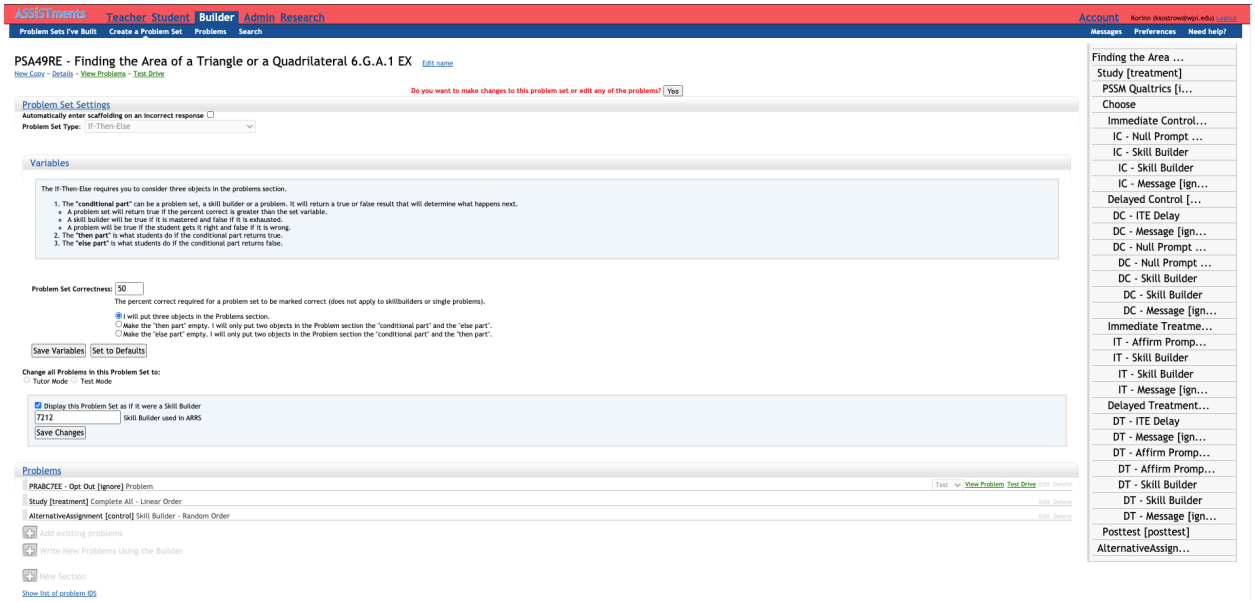


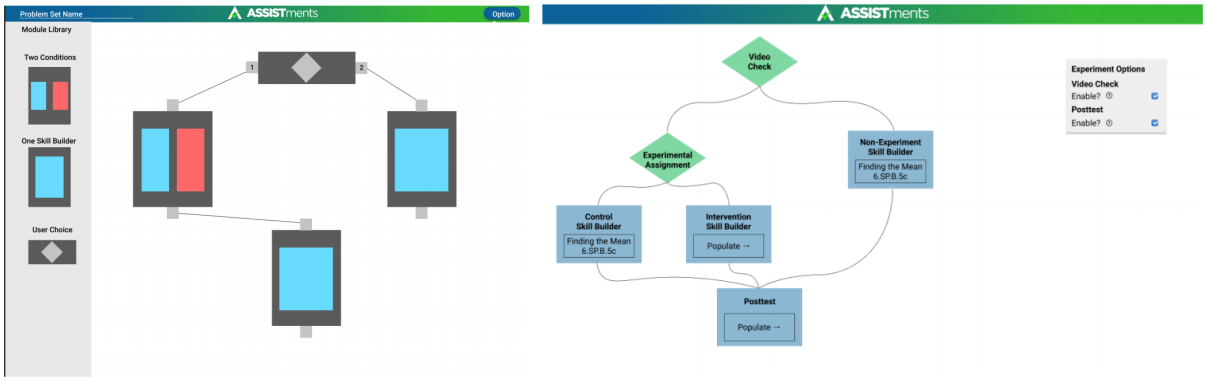

*Figure 1: The user interface of the 1.0 builder.*

In the 1.0 builder, studies are represented as tree structures that align with the internal representation of the problem set's structure in the ASSISTments platform. However, a researcher with little computer science experience might instead imagine their study as a flow chart logic model, thus making building their study in the 1.0 builder very difficult. Furthermore,

researchers that attempted to use the ASSISTments platform often had to meet with ASSISTments team members to get help building their studies. To increase the efficiency of researchers' and team members' time when making studies, the E-TRIALS team decided to recruit students for the first E-TRIALS IQP.

## The First IQP

The first E-TRIALS IQP, completed by Mr. Nicholas Krichevsky and Mr. Kamryn Spinelli under the guidance of Dr. Neil Heffernan, Dr. Korinn Ostrow, and Mr. Ryan Emberling, focused on redesigning the study building process from the ground up. First, they individually developed wireframe prototypes focused on guiding researchers through the study building process, including those depicted in Figure 2.




*(a) Mr. Krichevsky's prototype* *(b) Mr. Spinelli's prototype*

*Figure 2: Examples of prototype designs of E-TRIALS.*

They determined that the most important parts of study design to simplify for users were documentation, study visualization, and content selection, as these components were the most confusing in the 1.0 builder. Then the prototypes were unified into a single design by Mr. Emberling, in preparation for being implemented as a web application in the final term of the first IQP.

When a researcher loaded the final prototype, the application displayed a home page with studies that the researcher had created, as shown in Figure 3. From there, researchers could either edit their existing studies or create a new one by clicking the big plus button.

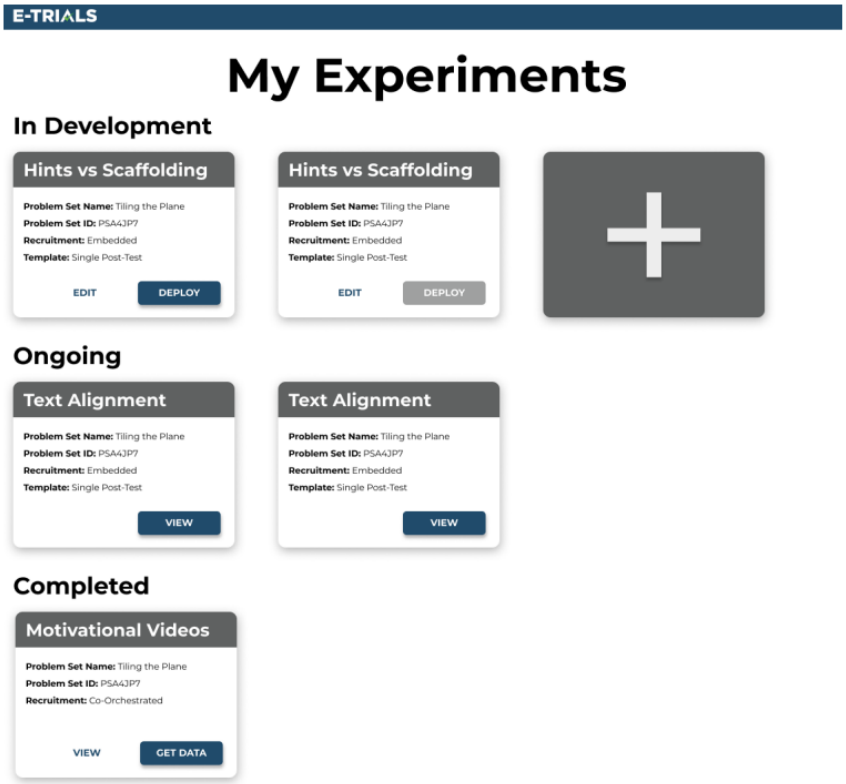



*Figure 3: The home page of the final prototype from the first IQP*

After creating the study, the researcher followed a predetermined page flow for filling in study information. First, the researcher input basic study information such as what the study name and research question were, and whether subjects would be prompted with the study directions or a media access verification question, as shown in Figure 4.

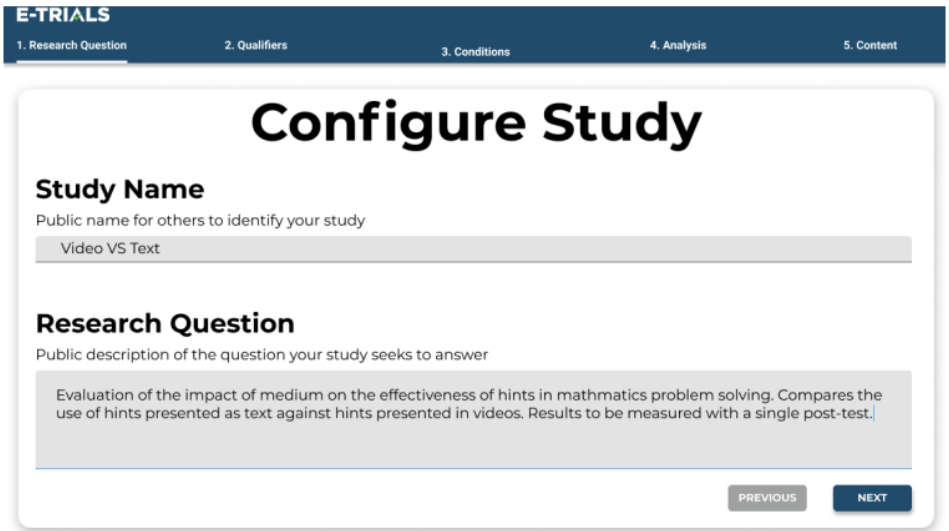



*Figure 4: The basic information page*

Then the researcher chose what assessments and conditions to add to the study, as shown in Figure 5. Importantly, the prototype included a preview panel which showed a responsive flowchart representation of the study that the researcher was building. This flowchart updated as the researcher filled in information, and maintained a strong visual of the flow of the study to keep the researcher informed.

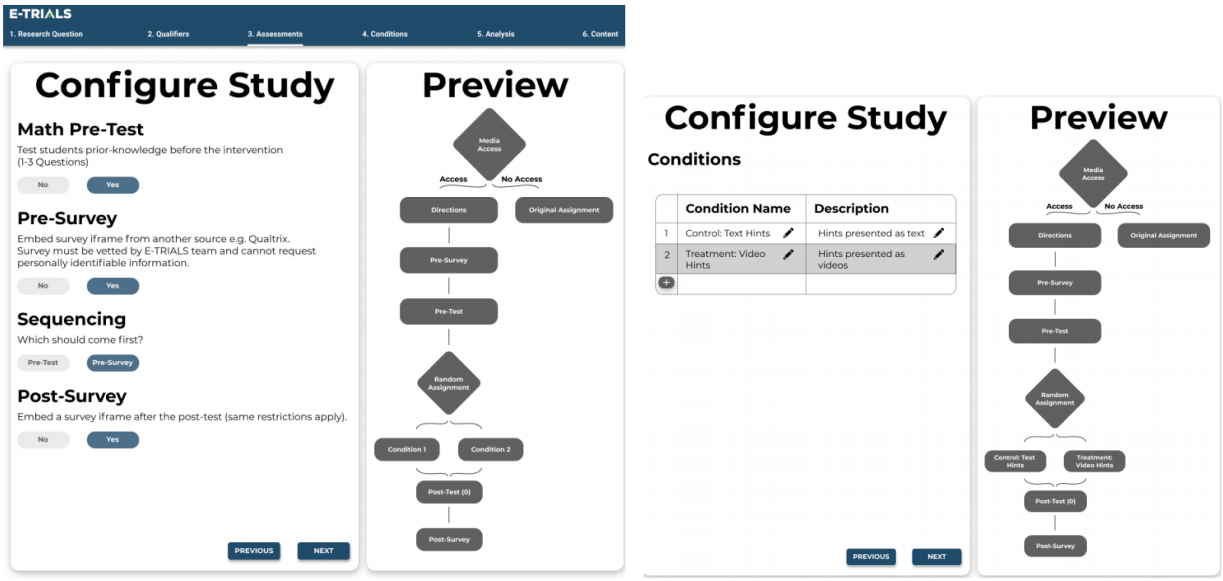



*Figure 5: The assessments and condition pages*

Next, the researcher had to configure their data analysis settings, shown in Figure 6. This determined the dependent variable of the study and what kinds of data the researcher received after the study had completed.

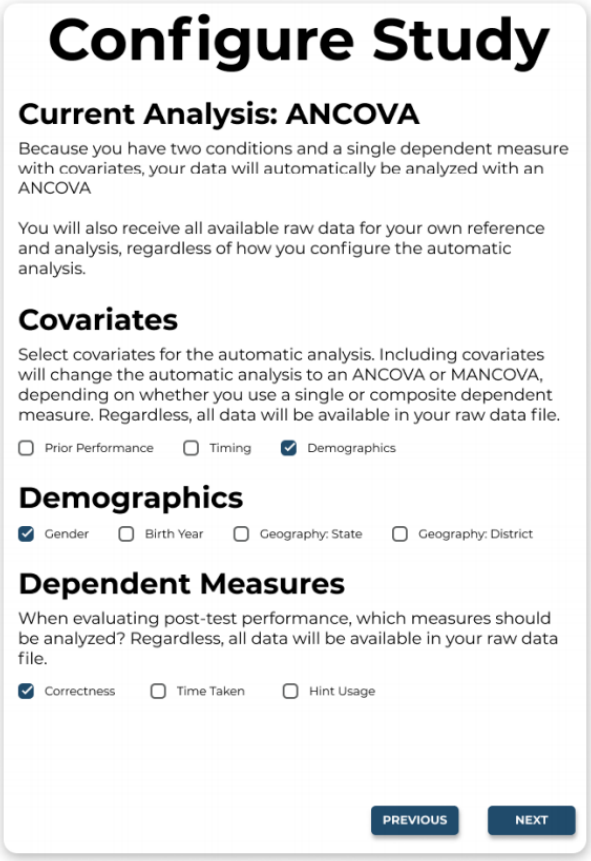



*Figure 6: The analysis panel*

Finally, the researcher selected content. To minimize the amount of information on screen, the content page displayed problem sets which could be refined by various filters, such as grade level, average problem correctness, and average problems to mastery, as shown in Figure 7. Once a problem set was selected, the researcher chose problems and edited them to match their previously selected experimental conditions, as shown in Figure 8. By clicking through the graph view, the researcher selected which problems were used for each component of the study.

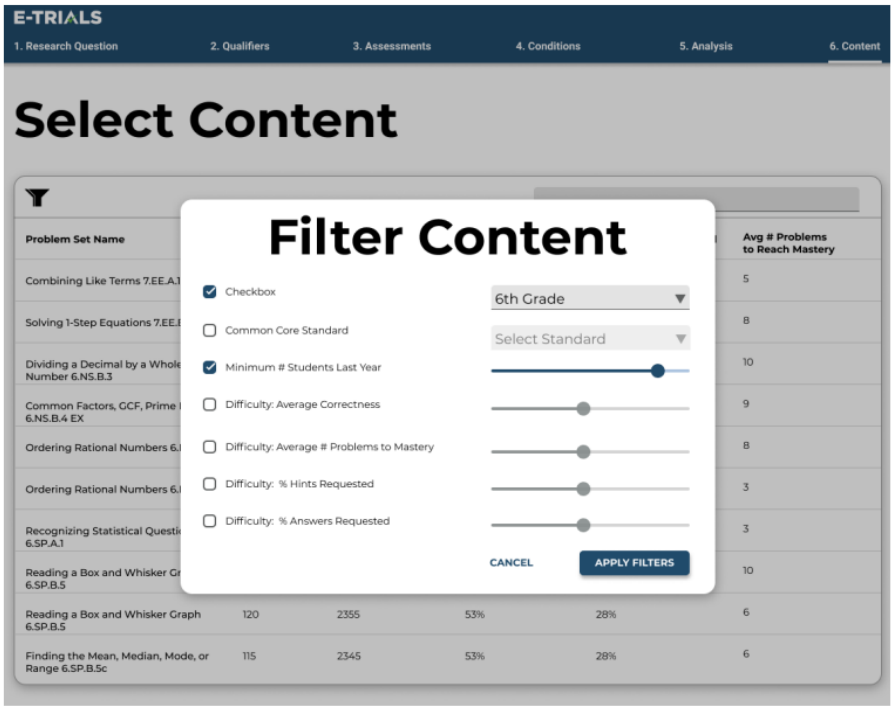
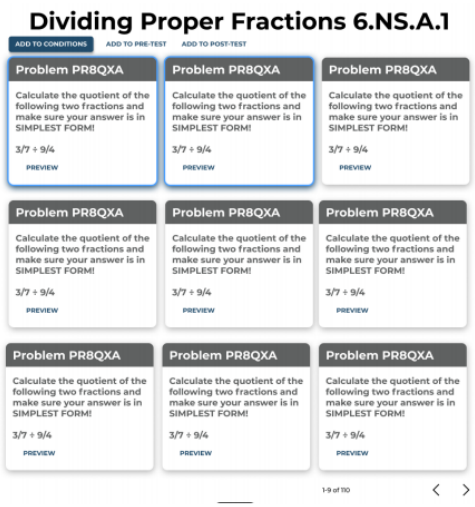
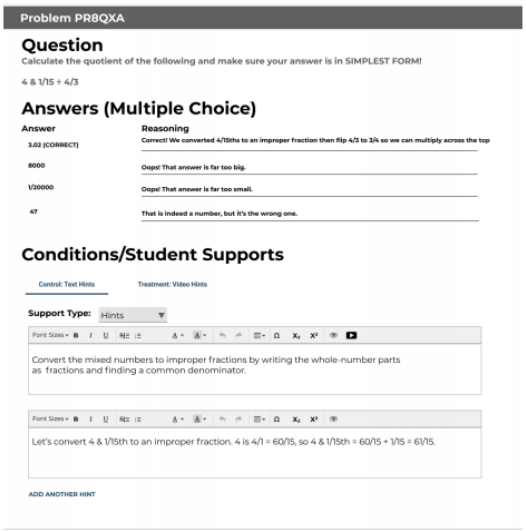



*Figure 7: the content selection page and content filter*




*(a) The problem selection screen*




*(b) The builder screen*

*Figure 8: Selecting and editing problems screens*

Mr. Krichevsky and Mr. Spinelli finished their IQP by beginning the development of the user interface. The resulting product was a user interface that closely resembled the final prototype in looks and design, but lacked essential functionality, including the ability to save studies and a user login system. At this point, the E-TRIALS team brought in Mr. Brian Rojas to begin developing the backend server and my IQP began to continue work on the user interface, as discussed in the remainder of this report.

# Development and Workflow

## Team Workflow

The E-TRIALS team developed the E-TRIALS application using the Agile Scrum methodology, where updates are delivered incrementally instead of all at once. Updates were delivered according to weekly development cycles, or "sprints." Mr. Emberling acted as the technical lead and ensured that each developer had tasks to complete for the sprint. Due to the COVID-19 pandemic, work performed on the E-TRIALS application was completed entirely online. On Mondays, the team held a sprint planning meeting and met over Zoom to discuss what tasks were going to be completed that week. To keep track of the task backlog, the team used Trello, a whiteboard-based collaboration tool shown in Figure 9 (Trello, n.d.). Trello helped the team organize development process tasks by stage, log who was working on each task, and prioritize the most important tasks. Agile Scrum also involves meeting everyday for about fifteen minutes to discuss daily updates and potential problems. Usually, team members perform this meeting standing up to make sure it is quick, and it is thus called a standup meeting. However, instead of meeting everyday, the E-TRIALS team used Slack, a professional channel-based messaging platform (Slack Team, n.d.), to communicate their tasks for the day in a channel called "etrials_standup."

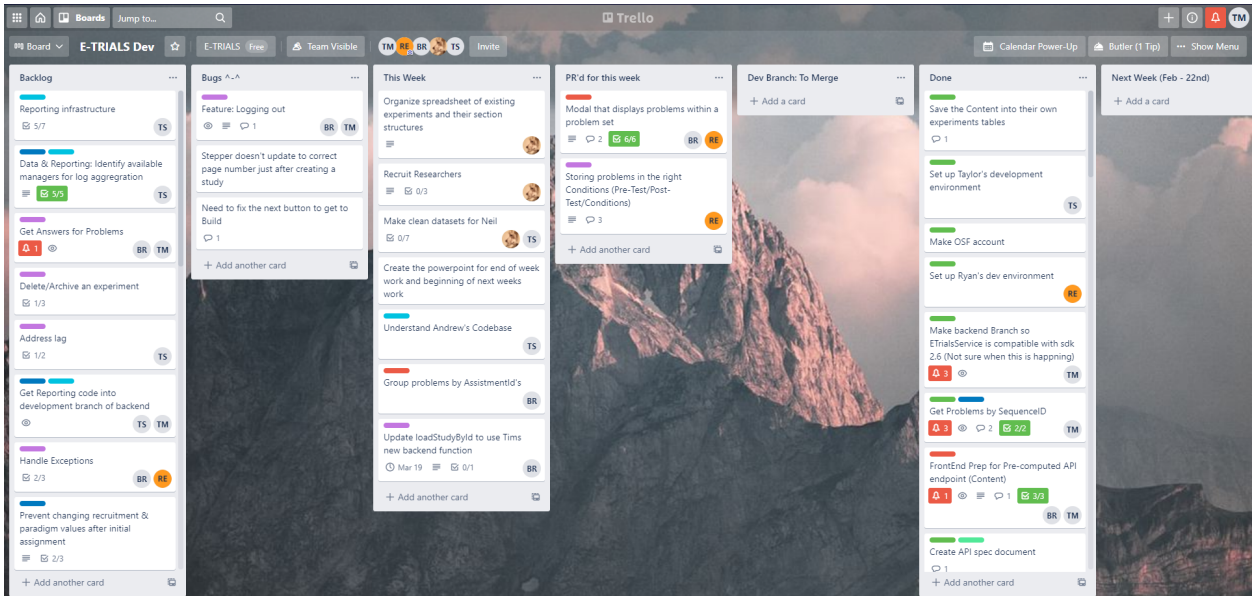



*Figure 9: The layout of the E-TRIALS Trello board*

## The E-TRIALS Stack

The E-TRIALS application is divided into two major pieces: the frontend and backend. The frontend, or the user interface, is written in VueJS, a reactive Javascript framework for web applications (Vue.js Team, 2020). Since the previous IQP resulted in a frontend written in Vue, the team decided that it was easiest to continue working in the same programming language instead of starting from scratch again. When started up, the frontend is run by a local NPM (Node Package Manager) server, which handles hosting and maintains Javascript packages and dependencies (Node.js Contributors, n.d.). NPM also handles running tests on the frontend. Unit tests are written using the Jest framework (Facebook, 2020), and the end-to-end tests are written using the Cypress framework (Cypress.io, 2020). Unit tests focus on breaking down the application into small pieces and testing each piece, while end-to-end tests check all units of the application at the same time and ensure compatibility between them.

The backend, or the server that handles the information from the frontend, is written in Java to match the ASSISTments infrastructure. A local backend server can be launched using Tomcat, a server container application (Tyson, 2019), but the backend also gets deployed to one of WPI's hosts and functions as the main development server. The backend also interfaces with a PostgreSQL database to maintain user and study data. Finally, the team used Github for version control to maintain the separation of each developer's code. By breaking features into branches using Github, each developer could work on and test their feature without worrying about another feature that was in progress.

# Features

## Frontend-Backend Integration

### Goals

The primary goal of integrating the frontend and backend was to evolve the prototype into a more complete product. Prior to integration, the team developed the user interface and server separately. Due to the nature of web design, the user interface was subject to many iterations of redesigns and tweaks, so keeping it separate from the server allowed for numerous updates. However, study data was stored only on the frontend, and thus could not be accessed at different computers or even saved across multiple web-browsing sessions. By connecting the two pieces together, the application would be able to save data across sessions, so researchers could make a study, take a break, and resume working on the study.

A second benefit of integration was that the application could be tested to a greater extent. The prototype's tests dealt with navigating through the limited functionality of the separated user interface. However, fully integrated end-to-end tests could more comprehensively simulate user interactions and validate that the system was correctly integrated (What is end-to-end (e2e) testing?..., 2019). Furthermore, an integrated application would also help prepare for complex user testing. The original design was built and tested in Figma, a web-based prototyping tool (About Figma, n.d.). While it allowed for rapid iteration on the E-TRIALS design, it lacked certain functionality that made conducting thorough user tests with researchers difficult if not impossible. A fully integrated application would be significantly more interactive than the Figma prototype, and it could be deployed to a web server that researchers could access on their own computer, as in-person user testing was impossible during the COVID pandemic.

### Design

At the highest level, the E-TRIALS application adheres to the Model-View-Controller (MVC) architectural pattern. MVC allows for independent development of its parts, which speeds up the development process. In the case of E-TRIALS, the database functions as the 'model,' maintaining study and problem set data. The backend is the 'controller,' and it receives user input from the frontend and updates the database. Lastly, the frontend acts as the 'view,' and displays application data and the actions a user can take. When the user interacts with it, the frontend interprets the input and sends it to the backend. Dividing up these sections of code maintains the separation of concerns, where logic and data are organized and separated by function. This simplifies the code structure and decreases the amount of code that developers have to write since each feature is kept separate from the others (Viva, 2020).

The application also follows the Client-Server architecture, as it is a simple and cheap option to employ across many users. The frontend acts as the 'client' and is deployed to any user

that requests its webpage. It is independent from other clients, so researchers can separately build their own studies. When a researcher has made changes, the data can then be sent to the 'server.' The 'server' receives and stores data from all clients, so a researcher can access their studies from any computer. The Client-Server architecture further maintains the separation of concerns, as user-facing data is kept detached from server-side data and logic.

## Implementation

The process of integration was broken up into three phases to ensure a comprehensive transition from the user interface prototype to the full stack application. The slow development also allowed the developers to learn about the mechanisms necessary to complete integration, namely HTTP requesting. As the standard for most web-applications, HTTP (HyperText Transfer Protocol) is a simple requesting protocol for sending web data between a server and client.

Phase one was devoted to sending single requests to the backend from the test page in order to ensure that the backend correctly received the request and updated the database accordingly. In phase two, the team built mock application pages and ensured that each page in its entirety could properly interact with the backend. Finally, phase three was dedicated to fully connecting the application to the backend and rewriting end-to-end tests to check for the persistence of data across sessions in the database.

The frontend portion of the integration was built into the pre-existing UI (User Interface). While little of the UI was updated, many changes were made to the Javascript under the hood. Primarily, end points that accessed mock data were replaced with calls made by Axios, a node.js HTTP request library (Axios, 2020). To access and update data from the backend, the store creates a promise and then makes its request. The promise serves as a placeholder for a value while the request is asynchronously performed. When the request is completed, the promise is fulfilled and the value returned can be used in later execution (MDN contributors, 2021). After the promise is returned, the store updates its values, which in turn updates the user view. This process ensures that the data in the database, local frontend data, and what the user sees are all synced together.

To be able to switch between various backend servers, an environment configuration file was also added. The file allows the developer to launch the frontend server using different backend servers, and differentiates between using the mock data, a locally run backend, the development backend server, and the production backend server. Since the team used different servers when working on different parts of the application, such as user interface development versus end-to-end testing development, easily being able to switch between these servers improves productivity.

In order to integrate, the E-TRIALS team had to build a new backend. Written on top of the existing ASSISTments SDK (Software Development Kit) version 2.6, the backend serves as a RESTful (REpresentational State Transfer) API that connects the frontend to the ASSISTments database via HTTP requests. The backend is structured into a Model-View-Controller-Service architecture, and is thus broken into 4 parts. The controller classes serve as the 'view' and relay

data between the ‘service’ layer and the frontend. The data from the controllers is then sent to the manager classes, which function as the Service layer and deal with high level application logic and pass that data to the Controller layer. Functioning as the Controller layer, the domain access objects (DAOs) then directly update the database. Finally, the domain objects serve as the ‘model’ layer and represent the database objects in Java. An example of this structure can be seen in the Appendix.

Some additional features were also required to fully integrate the backend with the frontend. Prior to integration, any user could access any other users’ study data. Thus the team added user authentication to prevent unauthorized users from updating or even accessing anyone else’s study data. Unauthorized requests result in a “*403: Forbidden error.*” To support more error codes, the team also added an exception handling controller. This controller receives errors that occur on the server, resolves them, and then informs the user what the cause of error was. Finally, the team wrote custom database queries to complete the end-to-end tests. Because the testing process adds studies to the database on each run, the database frequently fills up with many test studies. Thus, to prevent the end-to-end tests from accessing incorrect study data, the tables are deleted completely before each run.

## Simplifying Studies

### Goals

At their base level, ASSISTments studies are very complicated. Because building studies require intimate knowledge of specific experiment language and advanced building features in ASSISTments, the E-TRIALS team wanted to simplify the process for researchers to increase the volume and reduce the cost (in staff support time) of studies conducted. For example, in the original 1.0 study builder, researchers had to nest complicated if-then-else structures to create a study flow. This made the process difficult and time consuming to complete, and almost always required the assistance of an E-TRIALS team member, even though - if given the right environment - researchers should be able to design and implement a study on their own. Thus, the team wanted to tailor E-TRIALS to the way researchers’ approach problems by incorporating the ability to toggle small parts of a study. These modules would give researchers the freedom to design their study the way they wanted while still concretely guiding them through the study making process.

### Design

A major part of the design involved translating frontend data structures into database tables, as the mockup was not yet connected to a database. Since the objects on the frontend were simple, the changes needed to translate them were small. For example, every study has a set of conditions, so the team moved the condition object to its own table and connected it to its study with a foreign key. This relationship is represented as the arrow that points from the ‘condition’

entity to the ‘study’ entity and goes through the ‘has’ relation in Figure 10. The same was true for a study’s contributors, so a contributor table was also built.

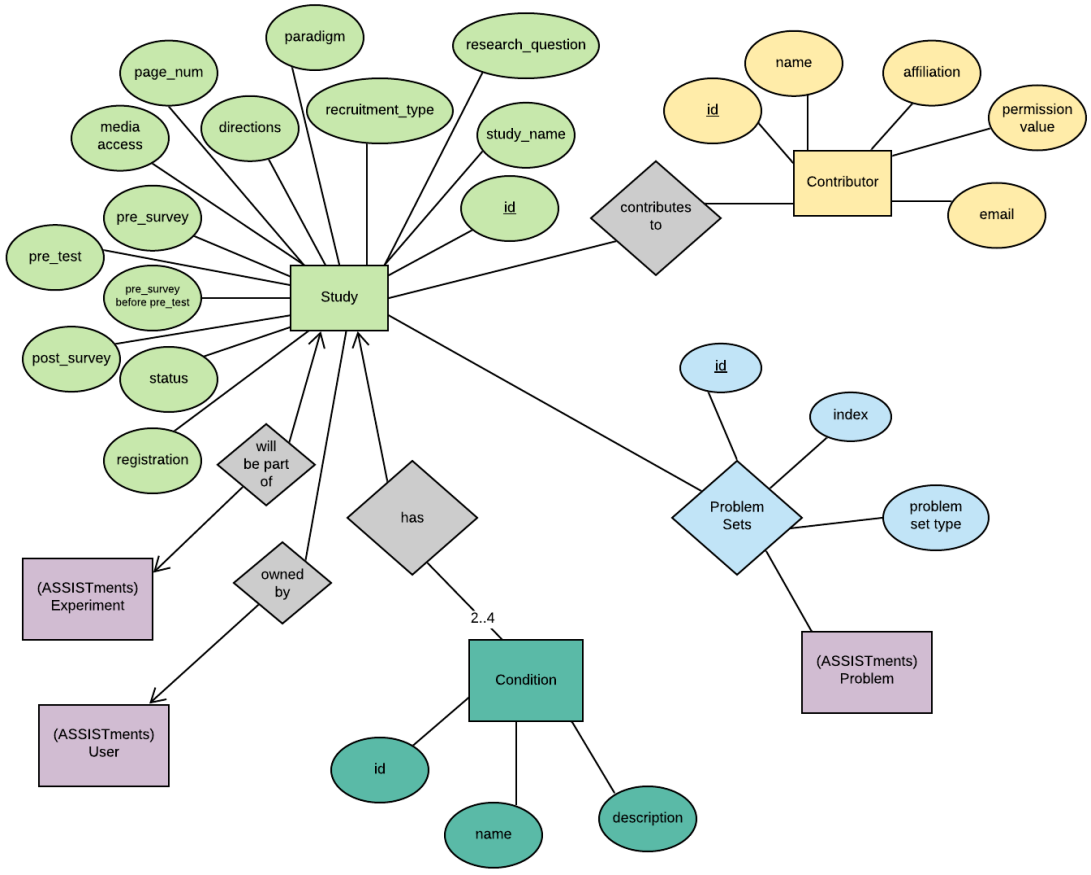



*Figure 10: the Entity Relationship Diagram of the E-TRIALS tables*

Another issue that had to be factored into the database design was associating problems with each study. To decrease the complexity of the tables, any problem that's associated with the study is put in the same table called ‘WIP problem set.’ However, each study needs to have different divisions of problems. To differentiate between parts of the study, i.e. pretest, post test, and experimental conditions, an extra column was added to the table to distinguish between these problem set types. This way, the relationship between problems and studies was simplified into one table, but the problem set type column could differentiate between problems that were part of conditions and those that were part of assessments.

Another challenge when dealing with these problems was the need to maintain the order of the problems. In some types of studies the order of the problems might be important to the

researcher, so the team needed to account for that. Many suggestions were made including ordering the problems using tree structures, singly-linked lists, and doubly-linked lists, each with their own benefits and costs. For example, the linked lists worked well for maintaining problem order, as one problem pointed to the next in the list, but accessing the whole list would take a long time, because each problem would have to be accessed one after the other. In the end, the team decided to use an index column, where the index represented the problem's position in the order of the study. As the simplest option, indexing maintains clarity and flexibility, and it also could be ignored when querying the database because the frontend would order the problems instead, saving time on accessing the database.

## Implementation

Following this design, the team went on to implement the E-TRIALS database. ASSISTments uses Table Definition Files (TDFs) to automatically generate database tables and code associated with the database. Instead of writing each Structured Query Language (SQL) query explicitly, TDFs organize table data into explicit sections, increasing the file's readability. On top of that, when tables need to be changed, a developer can easily update a line in the TDF, whereas a query may need to be rewritten entirely to properly update the table. Figure 11 demonstrates the difference in clarity between a TDF and SQL query that accomplish the same task.

```
[info]
table=conditions
domainObject=Condition
tableTypeEnum=org.assistments.etrials.dao.impl.EtrialsTableType
isLegacyTable=false

[fields]
id = pk
name = text
description = text
psid = integer
study_id = integer

[canBeNull]
description

[foreignKeys]
study_id = studies(id)
```

*(a) The Table Definition File that generates the conditions table*

```
drop table conditions;

create table conditions {
    id int not null;
    name text not null;
    description text;
    psid int not null;
    study_id int not null;
    constraint conditions_pk primary key id;
    constraint conditions_studies_fk foreign key study_id references studies(id);
}
```

*(b) The SQL query that generates the conditions table*

*Figure 11: A TDF and SQL query that generate the same database table*

ASSISTments also uses the Spring framework to handle a lot of low level business logic when developing enterprise applications, including setting up database connections and maintaining proper SQL syntax when querying and updating the database (Spring Team, n.d.). In the case of E-TRIALS, Spring serves as the connection between the Java API and the database, and can generate tables in the database when they are not present at connection, which has saved the team a lot of time in development and maintenance.

# Content Selection

## Goals

The process of selecting problems for a study is a central part of E-TRIALS, and ensuring that researchers could quickly and easily pick content was critical. It was also important that the process for associating content with parts of the study was clear and robust, leaving no space for uncertainty. In particular, the team wanted to organize the content selection process to be more contiguous. For example, in the previous IQP's design, researchers were supposed to add a post-test to their experiment, work on another part of the study, and then add problems to the post-test. The separation of these two related tasks led to a disconnect in researchers' train of thought when assigning content to tests.

Furthermore, the graph of the study was a simple depiction of the study flow, but also functioned as an interface for adding problems to the study. The dual use of the graph was jarring, and did not support the "flow." Finally, the team also wanted to future-proof the design flow for future features. In particular, the team is making plans to add support for more complex types of experiments to E-TRIALS, but as there is a deadline for the minimum viable product, it has been tabled for now.

## Design

The new design for content selection rests on the new page flow. After some basic metadata is added, researchers add experimental conditions and then go straight to adding problems to those conditions. This connects the action of making conditions to the content that goes into them and keeps the process grounded. After some conditions are added, the researcher then adds their assessment modules and fills them with assessment content. Selecting assessments was moved after selecting conditions because choosing problems for conditions helps researchers become familiarized with the constraints of E-TRIALS experiments. With these constraints in mind, researchers can base their assessments on the conditions that they have already made.

Another important part of the design was the content selector itself. Instead of being its own page, problem sets now appear in a fullscreen modal when selecting content for a condition

or assessment. Like in the prototype design, problem sets are laid out in a table and showing researcher-relevant metadata. Problem sets can also be filtered by using the filters on the right bar of the modal. Once a researcher selects a problem set, they can click directly on it and a smaller modal pops up, revealing all of the problems in that problem set. This approach, shown in Figure 12, allows the researcher to focus on small pieces of content while simultaneously giving them the freedom to explore as much content as they can.

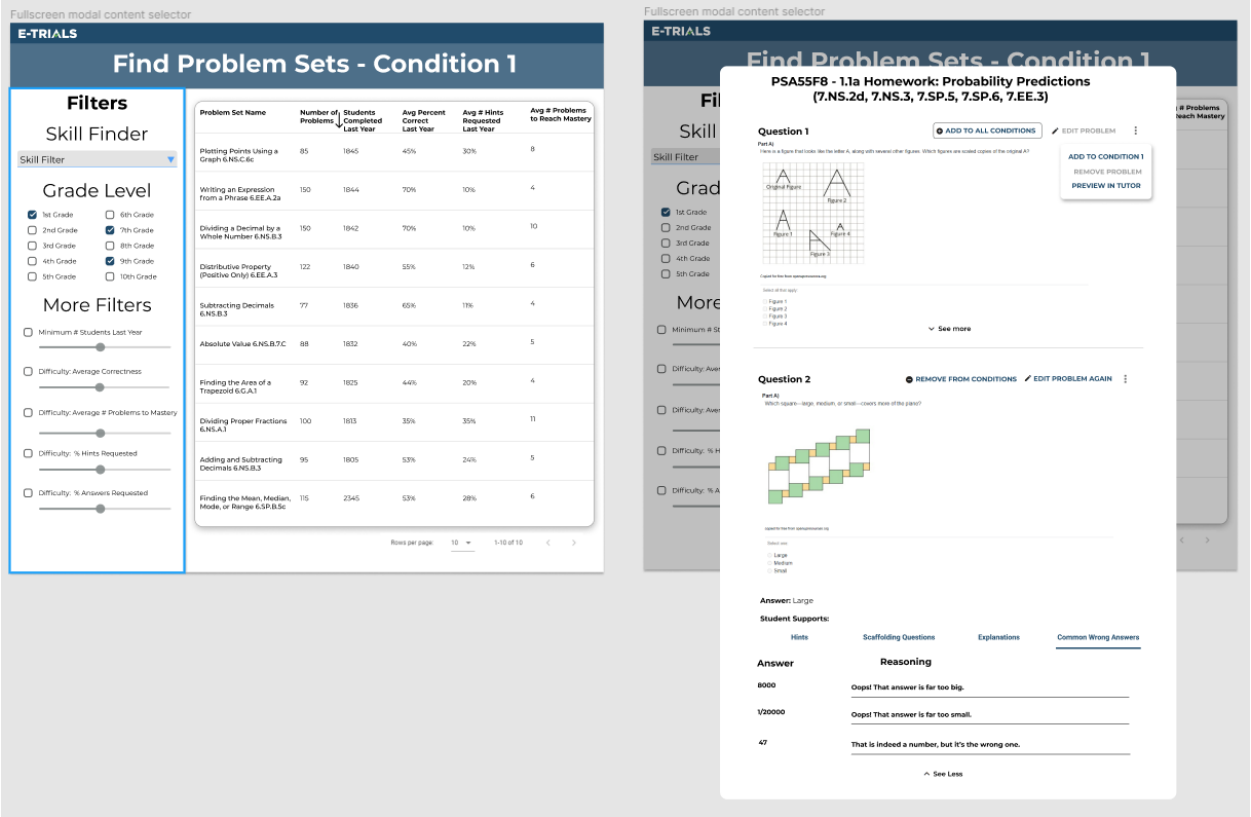



*Figure 12: The content selection modals as designed in Figma*

## Implementation

One important part of updating the page flow was fetching and displaying problem set data. While the ASSISTments SDK had the functionality to return information about problem sets, the team wanted to display aggregate data about each problem set in order to help researchers understand which problem sets would best suit their experiments. For example, researchers can select content based on the number of students that successfully completed a problem set in the last year. However, calculating this data was problematic. A single calculation on all problem sets took more than a full minute to return its results. Thus, all calculations on all problem sets would take tens of minutes, which was unacceptable for a reactive web application. To combat this, the team decided to run all of the calculations beforehand and add it to a separate database. This way, data can be requested and returned instantly, as the server does not need to

perform costly calculations. While updating this aggregate data table will still be very slow, updates only need to be performed sparsely. The E-TRIALS team is currently planning on updating this table every six months.

# Results

On its face, the current state of the E-TRIALS application closely mirrors the previous IQP's design. However, when the researcher first loads the application, they are met with the login screen shown in Figure 13. From there they can login to their ASSISTments account using their email.


*Figure 13: The login screen of the application*

After logging in, the researcher is presented with the home screen shown in Figure 14, where they can view studies that are in development, ongoing, and completed. From here, the researcher can create a new study, edit existing work, or deploy a finalized study.

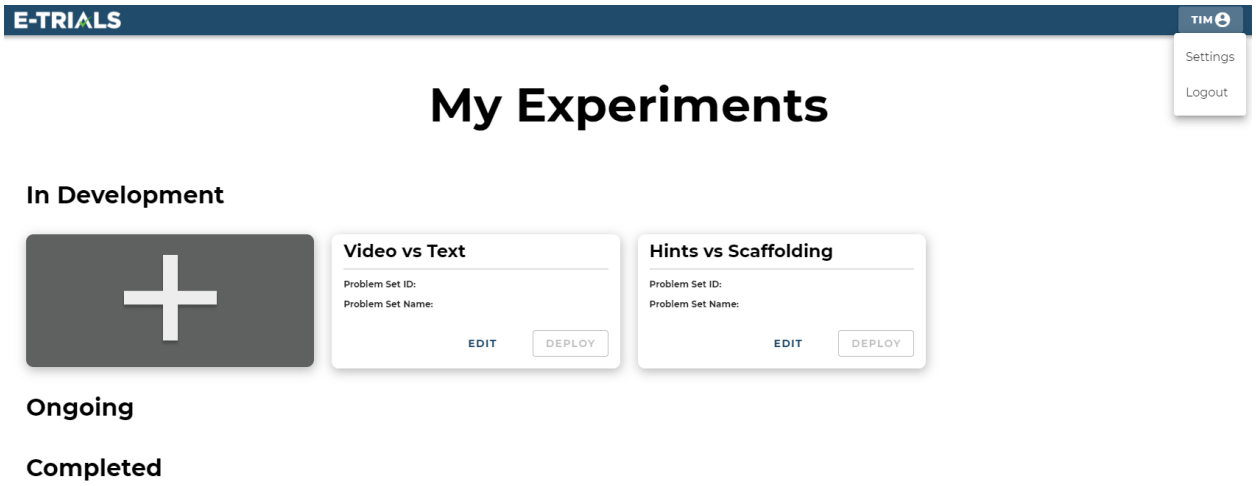



*Figure 14: The home screen with the user menu in the top right corner*

If the researcher decides to create a new study, they can click on the big plus button and the study creation modal pops up as shown in Figure 15. The researcher then fills in the study name and research question and clicks the continue button. This begins the study building process.

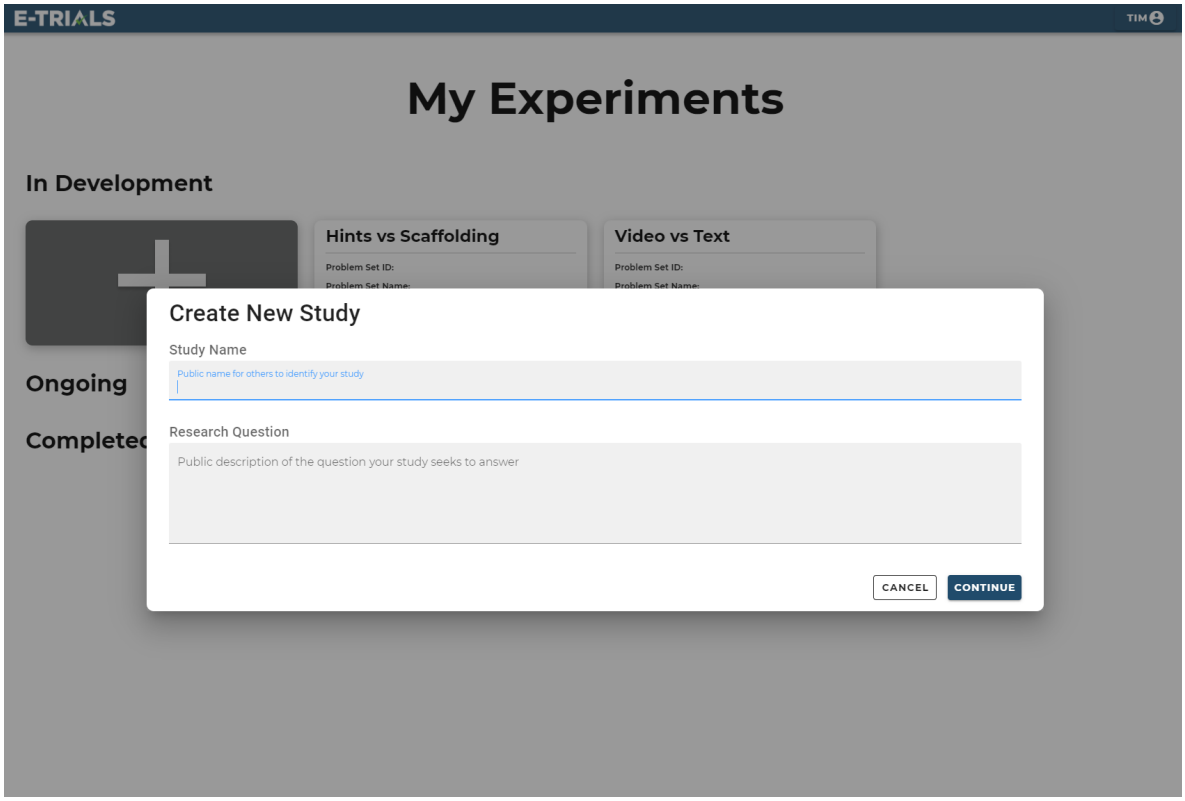
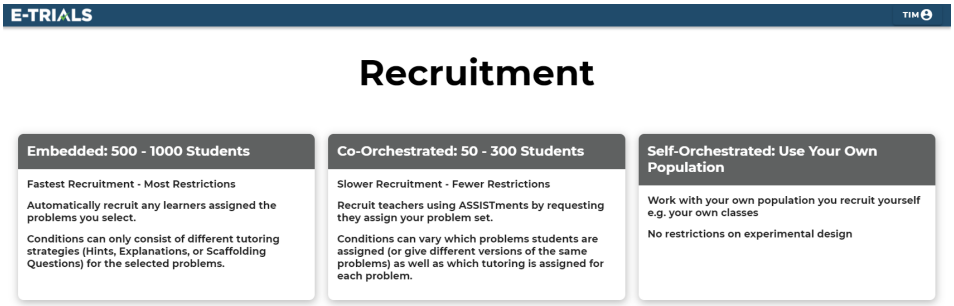
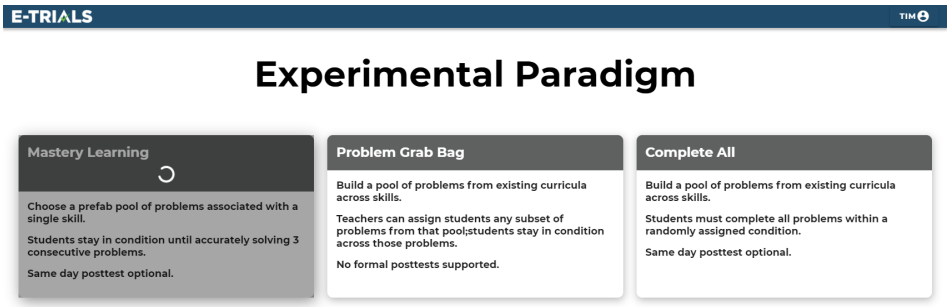



*Figure 15: The study creation modal*

Just like in the prototype design, the next step in the process is to select the recruitment type and experimental paradigm, as shown in Figure 16. Also in the experimental paradigm page in Figure 16 is an example of a loading icon. When a button is clicked, this icon appears to show that the researcher's input is being processed and will be finished soon.




*(a) The recruitment page*




*(b) The experimental paradigm page with a loading icon*

*Figure 16: The recruitment page and paradigm pages*

Once the basic study information has been filled in, the researcher can begin working on the substance of the study. Figures 17, 18, and 19 show the three main pages for assembling the flow of the study, where information changed in the Configure Study panel is immediately updated in the Preview panel. The researcher can jump to any part of the study by either clicking the page links in the study flow at the top of the application, they or can hit the navigation buttons to go to the next or previous page and follow the study creation flow.

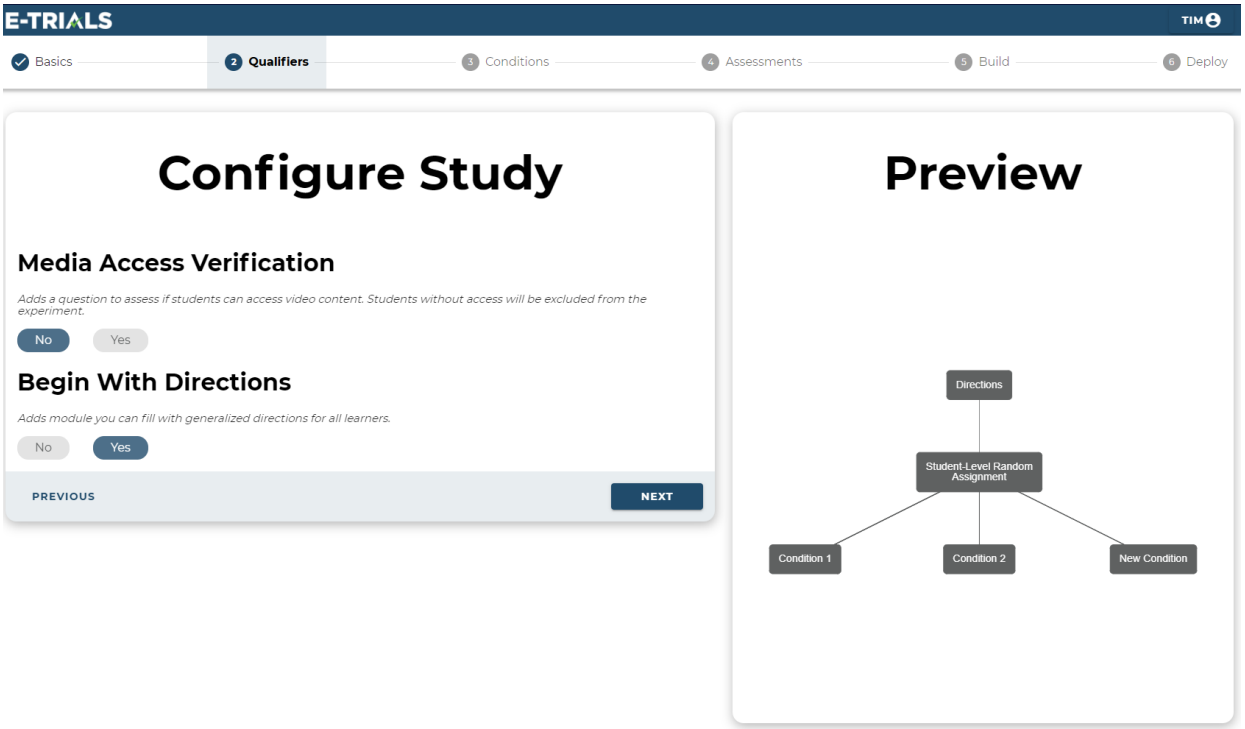



*Figure 17: The study qualifiers page with directions selected*

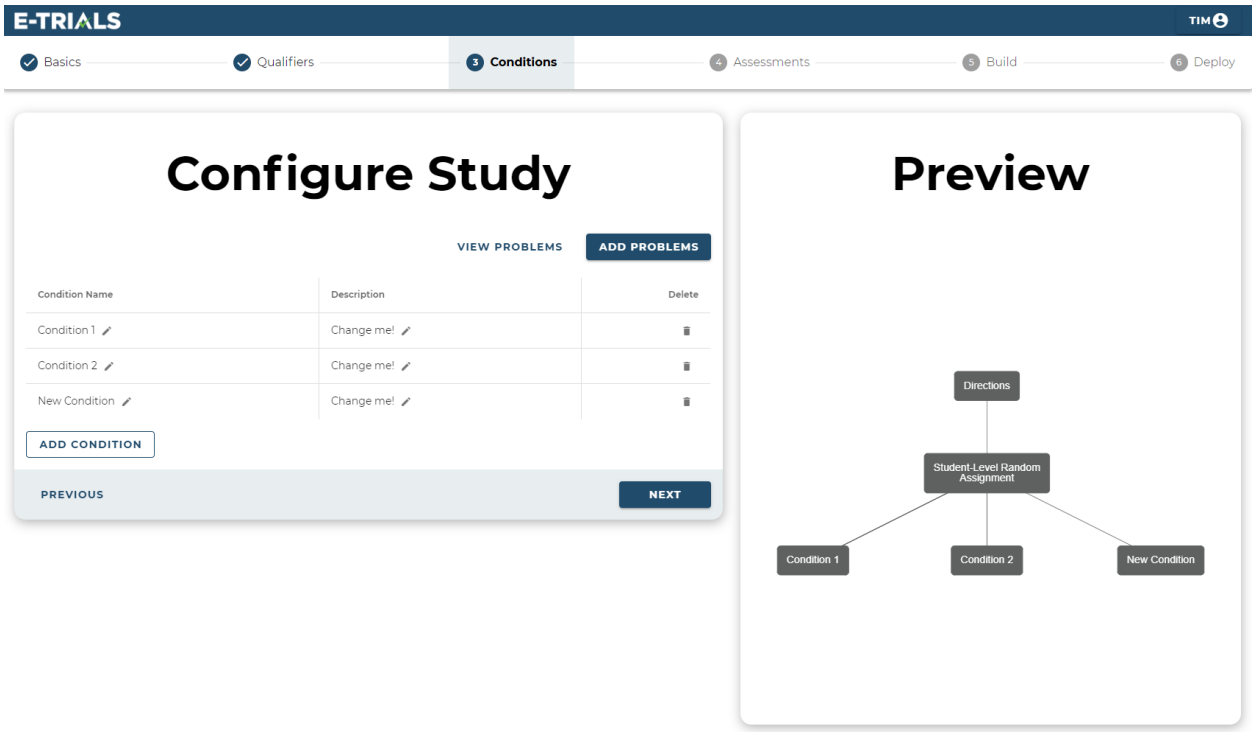



*Figure 18: The study conditions page*

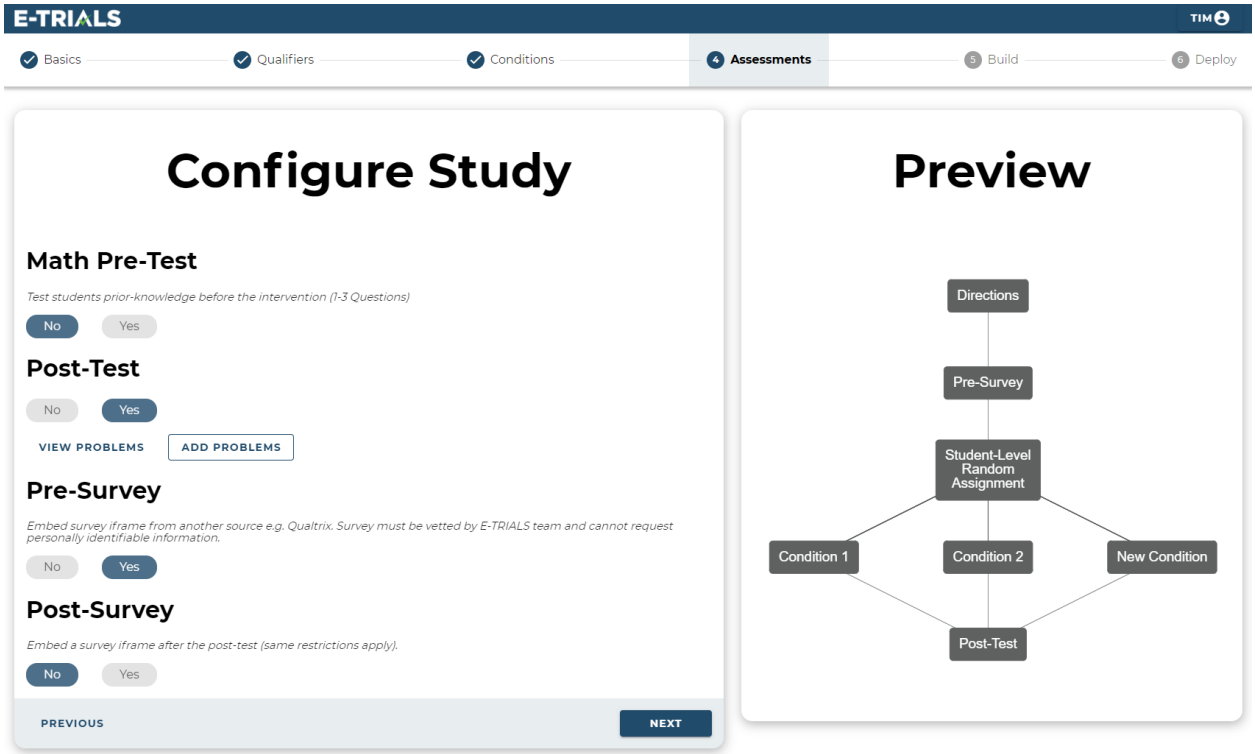



*Figure 19: The study assessments page*

From the Conditions or Assessment pages, the researcher can add problems to their study by clicking on the add problems button. This button opens up the full screen modal for problem set selection shown in Figure 20. Just like in the prototype design, the researcher can filter problem sets by various filters including grade level, average problem correctness, and average problems to mastery.

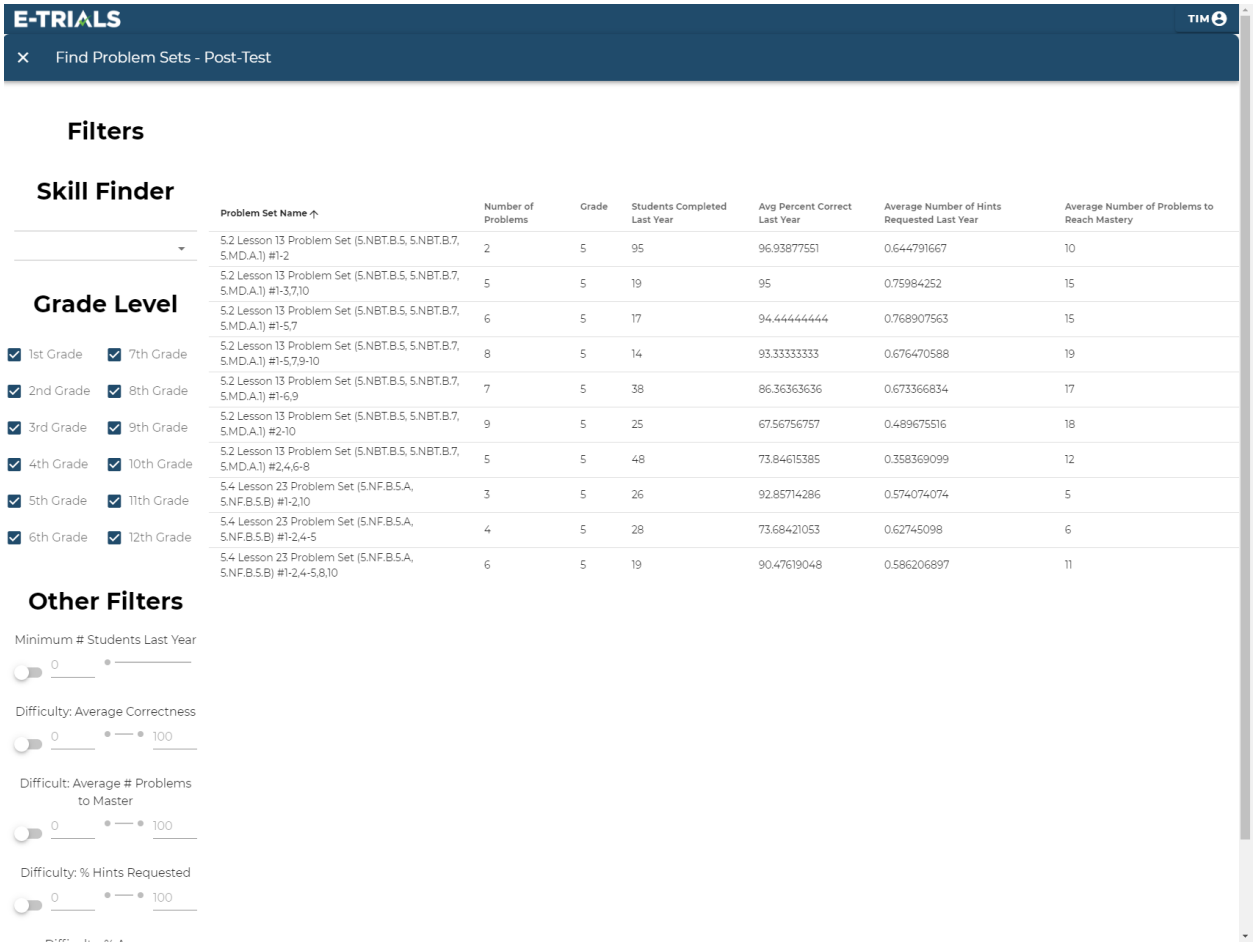



*Figure 20: The post test problem set modal*

When the researcher selects a problem set, the problem modal opens up, as shown in Figure 21. From here, the researcher can review each problem and choose whether or not to add it to the component of the study they are working on.

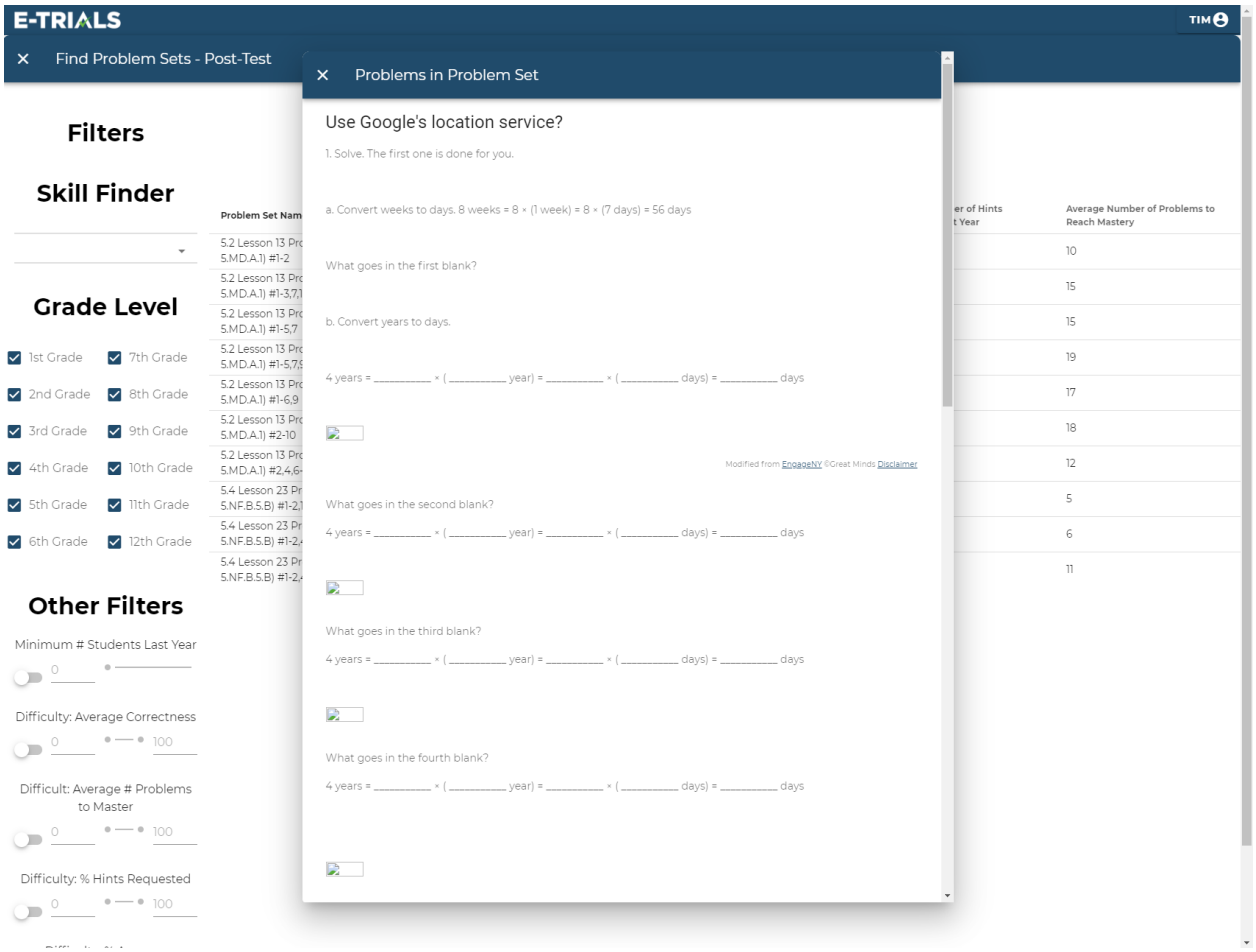



*Figure 21: The post test problem selector modal*

The build and deploy pages, as shown in Figure 22 and Figure 23, are also still in development. The current build page is left over from the previous IQP's application, but will become the page where researchers can edit problems to match their experimental conditions. Similarly, the deploy page will become the page where researchers can double check the structure of their study, register the study with Open Science Foundation (osf.io), and deploy their study into ASSISTments.

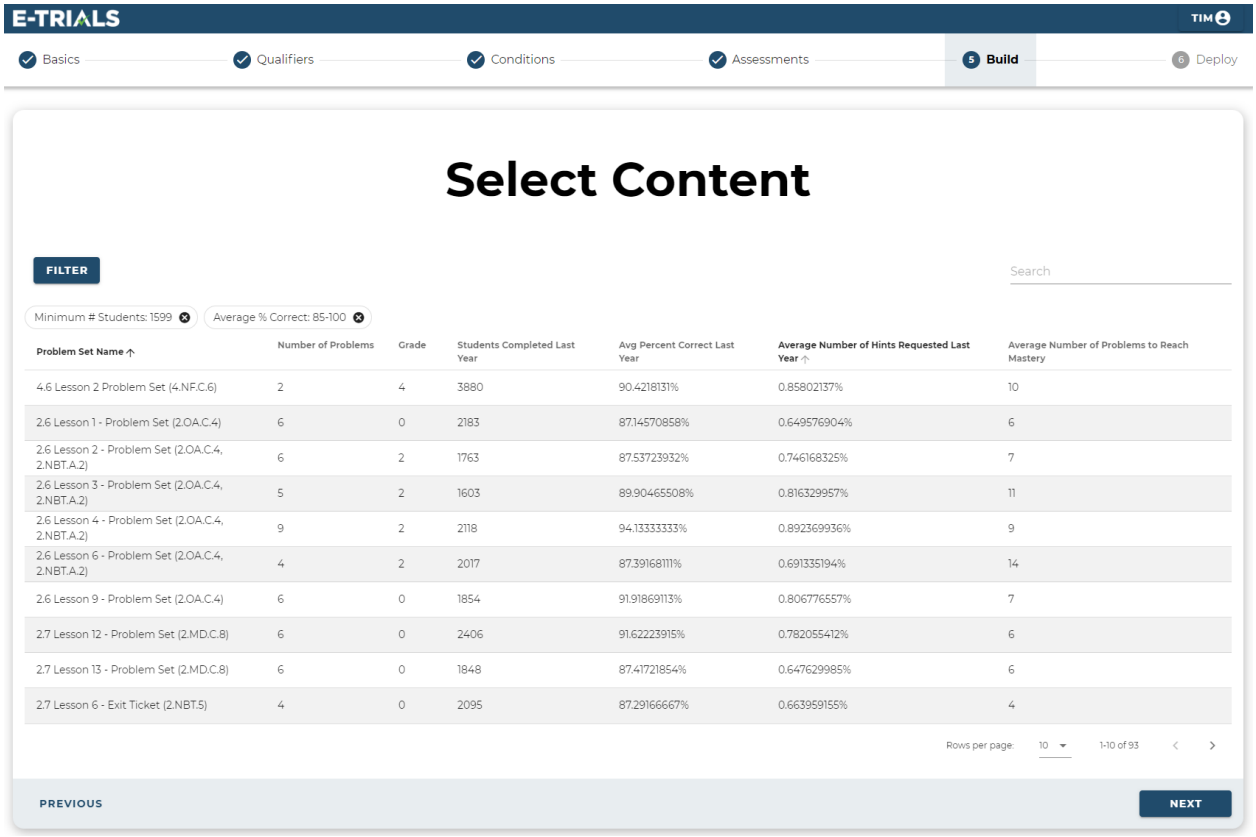



*Figure 22: The study content builder page*

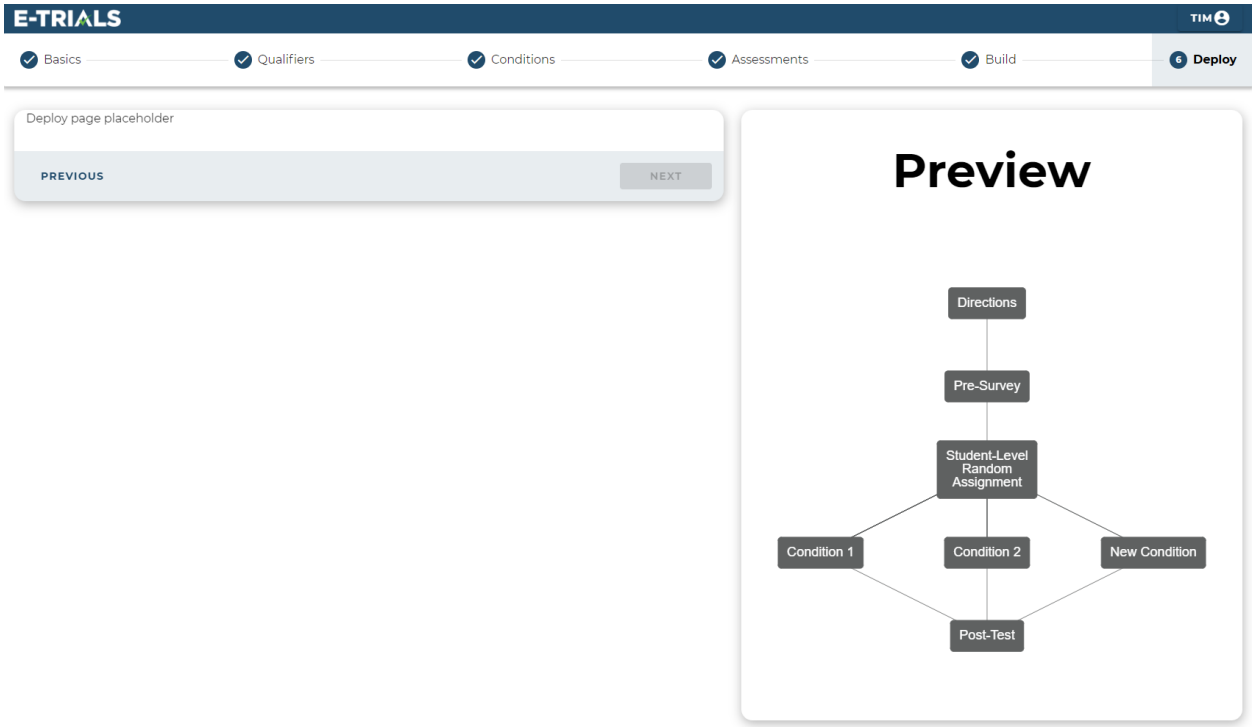



*Figure 23: The study deploy page*

# Discussion

## E-TRIALS and the Road Forward

Educational research can be very expensive, complicated, and bureaucratic. Researchers normally need to coordinate between students and teachers, and also have to submit their research plan before an Institutional Review Board (IRB), which is very time consuming. E-TRIALS abstracts much of that process and gets the researcher doing the important stuff: research. On top of that, due to the COVID-19 pandemic, more and more classrooms have turned to online learning. With so many students learning online - in environments with little regulation - it is imperative that researchers begin to assess the best methods for teaching in the 21st century. Further development of the E-TRIALS application will cause an influx of research and thereby produce outcomes that strengthen how material is taught online.

At this stage of E-TRIALS' development, the application is not ready to be released. There are two main features that still need to be implemented: the build and deploy pages. The build page is currently on hold because it needs to interface with the TeacherASSIST student support builder, which is still in development. The deploy page is also not finished because it requires study structures to be fully fleshed out before they can be deployed, and those structures depend on the ability to edit content, which takes place in the build page. Once these pages are complete, the team will be ready to launch the minimum viable product to the world.

As outlined in the previous IQP, researchers commonly collaborate when working on projects. Thus it would strongly improve the user experience if E-TRIALS included a collaboration feature, where multiple accounts could access and edit the same study. There has already been some development of the data structures needed to accomplish this, namely the collaborators table in the database, but the feature has not yet been implemented in the application.

One final thing the team could work on would be user testing. Once again, due the COVID-19 pandemic, in person user testing was impossible, and online testing was also difficult as the application could not be deployed to a researcher's computer. Getting real researchers to use E-TRIALS in its current state and provide feedback would solidify design decisions that the team has made over the past year and give further insight on how to optimize the study creation process. Launch plans include the recruitment of a limited cohort of grant funded researchers who will participate in two distinct rounds of user testing and iterative design following workshops that will be held for training and networking.

## Final Thoughts

This IQP was particularly tough as I was the only undergraduate working on the project. Despite that, I cannot stress enough how thankful I am for Mr. Emberling and Dr. Ostrow's help through this whole process. Without them, I would not have made it this far. I am also thankful

for my developer in crime, Mr. Rojas, without whom I would not have been able to expand my developing horizons. I am fully confident in the team's ability to finish making E-TRIALS a one-of-a-kind application and wish them the best of luck.

# Appendix

## ContentController.java

```
package org.assistments.etrials.web;

@RestController
@RequestMapping("/content")
public class ContentController
{
  @Autowired
  private EtrialsContentManager cntMger;

  @NeedsAuthority("USER")
  @RequestMapping(value = "/getProblemSetData", method = RequestMethod.GET,
    produces = MediaType.APPLICATION_JSON_VALUE)
  public List<Content> getContent() throws NotFoundException
  {
    return cntMger.getProblemSetData();
  }

  @NeedsAuthority("USER")
  @RequestMapping(value = "/getProblems", method = RequestMethod.POST, consumes =
MediaType.APPLICATION_JSON_VALUE,
    produces = MediaType.APPLICATION_JSON_VALUE)
  public List<ProblemRow> getProblems(@RequestBody IdClass sequenceWrapper) throws
NotFoundException
  {
    return cntMger.getProblemsFromSequenceId(sequenceWrapper.getId());
  }
}
```

## EtrialsContentManagerImpl.java

```
package org.assistments.etrials.manager;

@Component
public class EtrialsContentManagerImpl implements EtrialsContentManager
{
  @Autowired
  SequenceManager smgr;
  @Autowired
  ContentDao cntDao;

  @Autowired
  TutorProblemManager problemManager;
```

```
  @Autowired
  TutorProblemSetManager problemSetManager;

  public List<Content> getProblemSetData() throws NotFoundException
  {
    return cntDao.findAllObjects();
  }

  public List<ProblemRow> getProblemsFromSequenceId(int sequenceId) throws NotFoundException
  {
    return problemManager.getProblemDatasByAssistmentIds(problemSetManager

.findAllAssistmentsIdsByHeadSectionId(smgr.findSequenceBySequenceId(sequenceId).getHeadSecti
onId()));
  }
}
```

# Content.java

```
package org.assistments.etrials.domain;

/*
 * Generated code: Tim 2021-01-26T18:47:32.605Z
 */
public class Content extends DbObjectImpl
{
  private String name;
  private int grade;
  private int numberOfProblemsInProblemSet;
  private int numberOfStudentsThatCompletedProblemSetLastYear;
  private double averagePercentStudentsCompletedProblemSetLastYear;
  private double averageNumberOfHintsRequestedForProblemSetLastYear;
  private String commonCoreStandard;
  private int averageNumberOfProblemsCompletedToAchieveMastery;

  public Content()
  {}

  public void setName(String name)
  {
    this.name = name;
  }

  public String getName()
  {
    return this.name;
  }

  public void setGrade(int grade)
  {
    this.grade = grade;
```

```
  }

  public int getGrade()
  {
    return this.grade;
  }

  public void setNumberOfProblemsInProblemSet(int numberOfProblemsInProblemSet)
  {
    this.numberOfProblemsInProblemSet = numberOfProblemsInProblemSet;
  }

  public int getNumberOfProblemsInProblemSet()
  {
    return this.numberOfProblemsInProblemSet;
  }

  public void setNumberOfStudentsThatCompletedProblemSetLastYear(int
numberOfStudentsThatCompletedProblemSetLastYear)
  {
    this.numberOfStudentsThatCompletedProblemSetLastYear =
numberOfStudentsThatCompletedProblemSetLastYear;
  }

  public int getNumberOfStudentsThatCompletedProblemSetLastYear()
  {
    return this.numberOfStudentsThatCompletedProblemSetLastYear;
  }

  public void
    setAveragePercentStudentsCompletedProblemSetLastYear(double
averagePercentStudentsCompletedProblemSetLastYear)
  {
    this.averagePercentStudentsCompletedProblemSetLastYear =
averagePercentStudentsCompletedProblemSetLastYear;
  }

  public double getAveragePercentStudentsCompletedProblemSetLastYear()
  {
    return this.averagePercentStudentsCompletedProblemSetLastYear;
  }

  public void
    setAverageNumberOfHintsRequestedForProblemSetLastYear(double
averageNumberOfHintsRequestedForProblemSetLastYear)
  {
    this.averageNumberOfHintsRequestedForProblemSetLastYear =
averageNumberOfHintsRequestedForProblemSetLastYear;
  }

  public double getAverageNumberOfHintsRequestedForProblemSetLastYear()
  {
```

```
    return this.averageNumberOfHintsRequestedForProblemSetLastYear;
  }

  public void setCommonCoreStandard(String commonCoreStandard)
  {
    this.commonCoreStandard = commonCoreStandard;
  }

  public String getCommonCoreStandard()
  {
    return this.commonCoreStandard;
  }

  public void setAverageNumberOfProblemsCompletedToAchieveMastery(int
averageNumberOfProblemsCompletedToAchieveMastery)
  {
    this.averageNumberOfProblemsCompletedToAchieveMastery =
averageNumberOfProblemsCompletedToAchieveMastery;
  }

  public int getAverageNumberOfProblemsCompletedToAchieveMastery()
  {
    return this.averageNumberOfProblemsCompletedToAchieveMastery;
  }

  @Override
  public String toString()
  {
    StringBuilder sb = new StringBuilder(Content.class.getSimpleName())
      .append(":").append(Util.NL)
      .append(ToStringHelper.variableToString("id", super.getId(), true))
      .append(ToStringHelper.variableToString("name", name, true))
      .append(ToStringHelper.variableToString("grade", grade, true))
      .append(ToStringHelper.variableToString("numberOfProblemsInProblemSet",
numberOfProblemsInProblemSet, true))

.append(ToStringHelper.variableToString("numberOfStudentsThatCompletedProblemSetLastYear",
        numberOfStudentsThatCompletedProblemSetLastYear, true))

.append(ToStringHelper.variableToString("averagePercentStudentsCompletedProblemSetLastYear",
        averagePercentStudentsCompletedProblemSetLastYear, true))

.append(ToStringHelper.variableToString("averageNumberOfHintsRequestedForProblemSetLastYear"
,
        averageNumberOfHintsRequestedForProblemSetLastYear, true))
      .append(ToStringHelper.variableToString("commonCoreStandard", commonCoreStandard,
true))

.append(ToStringHelper.variableToString("averageNumberOfProblemsCompletedToAchieveMastery",
        averageNumberOfProblemsCompletedToAchieveMastery, true));
    return sb.toString();
  }
```

```
}
```

## ContentDao.java

```
package org.assistments.etrials.dao;
/*
 * Generated code: Tim 2021-01-26T18:47:32.594Z
 */
public interface ContentDao extends CommonDao<Content>
{
  enum Field implements DaoField
  {
    ID(DbDataType.PK,
      FieldModifier.PRIMARY_KEY, FieldModifier.REQUIRED),

    NAME(DbDataType.TEXT,
      FieldModifier.REQUIRED),

    GRADE(DbDataType.INTEGER,
      FieldModifier.OPTIONAL),

    NUMBER_OF_PROBLEMS_IN_PROBLEM_SET(DbDataType.INTEGER,
      FieldModifier.REQUIRED),

    NUMBER_OF_STUDENTS_THAT_COMPLETED_PROBLEM_SET_LAST_YEAR(DbDataType.INTEGER,
      FieldModifier.REQUIRED),

    AVERAGE_PERCENT_STUDENTS_COMPLETED_PROBLEM_SET_LAST_YEAR(DbDataType.DOUBLE,
      FieldModifier.REQUIRED),

    AVERAGE_NUMBER_OF_HINTS_REQUESTED_FOR_PROBLEM_SET_LAST_YEAR(DbDataType.DOUBLE,
      FieldModifier.REQUIRED),

    COMMON_CORE_STANDARD(DbDataType.TEXT,
      FieldModifier.OPTIONAL),

    AVERAGE_NUMBER_OF_PROBLEMS_COMPLETED_TO_ACHIEVE_MASTERY(DbDataType.INTEGER,
      FieldModifier.OPTIONAL);

    private DbDataType dataType;
    private List<Pair<FieldModifier, String>> modifierPairs;
    public String name;
    public FieldModifier[] modifiers;

    private static final List<Pair<FieldModifier, String>> noModifierPairs =
      new ArrayList<Pair<FieldModifier, String>>();

    Field(DbDataType dataType, FieldModifier... modifiers)
    {
```

```
      this.dataType = dataType;
      this.modifiers = modifiers;
      this.name = this.name().toLowerCase();
    }

    Field(DbDataType dataType, List<Pair<FieldModifier, String>> modifierPairs,
FieldModifier... modifiers)
    {
      this(dataType, modifiers);
      this.modifierPairs = modifierPairs;
    }

    @Override
    public String getName()
    {
      return this.name;
    }

    @Override
    public DbDataType getDbDataType()
    {
      return this.dataType;
    }

    @Override
    public FieldModifier[] getModifiers()
    {
      return this.modifiers;
    }

    @Override
    public List<Pair<FieldModifier, String>> getFieldModifierPairs()
    {
      if (this.modifierPairs == null)
      {
        return noModifierPairs;
      }

      return this.modifierPairs;
    }

    @Override
    public QueryTerm getQueryTerm(Object value)
    {
      return new QueryTerm(this.name, value);
    }

    @Override
    public QueryTerm getQueryTerm(RelationalOpType op, Object value)
    {
      return new QueryTerm(this.name, op, value);
    }
```

```
    @Override
    public NVPair getNVPair(Object value)
    {
      return new NVPair(this.name, value);
    }
  }

  static final String TableName = "content";

  static final Set<String> MultiFieldConstraints = new HashSet<String>(Arrays.asList(

  ));

  static final Set<Pair<String, String>> TableConstraints = new HashSet<>(Arrays.asList(

  ));

  static final Set<String> AdditionalSQLs = new HashSet<String>(Arrays.asList(

  ));

  static final Set<DaoField> UserXids = new HashSet<DaoField>(Arrays.asList(

  ));
}
```

# Works cited

6 advantages of using MVC model for effective web application development. (n.d.). Retrieved March 15, 2021, from https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/

About Figma, the collaborative interface design tool. (n.d.). Retrieved February 25, 2021, from https://www.figma.com/about/

Axios. (2020, December 1). Retrieved March 18, 2021, from https://www.npmjs.com/package/axios

Burnett, A. (2020, November 29). Immediate feedback. Retrieved March 5, 2021, from https://www.teachersgoinggradeless.com/blog/immediate-feedback

Cypress.io. (2020, May 5). Why cypress? Retrieved March 19, 2021, from https://docs.cypress.io/guides/overview/why-cypress.html.

Facebook. (2020). Jest – delightful javascript testing. Retrieved March 19, 2021, from https://jestjs.io/en/.

MDN contributors. (2021, March 16). Web technology for developers. Retrieved March 18, 2021, from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Node.js Contributors. (n.d.). *What is npm?* Node.js. Retrieved March 19, 2021, from https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/

Ostrow, K. S. (2015). *A multifaceted consideration of motivation and learning within assistments* (Unpublished master's thesis). Retrieved March 7, 2021, from https://digitalcommons.wpi.edu/etd-theses/1153

Ostrow, K. S. (2018). *A Foundation for Educational Research At Scale: Evolution and Application* (Unpublished doctoral dissertation). WPI. Retrieved March 1, 2021, from https://digital.wpi.edu/show/br86b377g

Ostrow, K. S. & Emberling, R. (2020, August 12). E-TRIALS: A web-based application for educational experimentation at scale. In the Learning at Scale workshop on Educational AB Testing at Scale.

Slack. (n.d.). What is Slack? Retrieved March 10, 2021, from https://slack.com/help/articles/115004071768-What-is-Slack-

Spring Team. (n.d.). Spring framework. Retrieved March 18, 2021, from https://spring.io/projects/spring-framework

Trello. (n.d.). What is trello? Retrieved March 10, 2021, from https://help.trello.com/article/708-what-is-trello

Tyson, M. (2019, December 19). What is Tomcat? The original Java servlet container. Retrieved March 19, 2021, from https://www.infoworld.com/article/3510460/what-is-apache-tomcat-the-original-java-servlet-container.html

Viva. (2020, December 21). 12 differences between object-oriented database and object-relation database. Retrieved March 18, 2021, from https://vivadifferences.com/12-differences-between-object-oriented-database-and-object-relational-database-plus-similarities/

Vue.js Team. (2020, February 24). Introduction – Vue.js. Retrieved March 10, 2021, from https://v3.vuejs.org/

What is end-to-end (e2e) testing? All you need to know. (2019, August 22). Retrieved March 5, 2021, from http://www.katalon.com/resources-center/blog/end-to-end-e2e-testing/#:~:text=The%20main%20purpose%20of%20End,for%20integration%20and%20data%20integrity