QuizASSIST: Hybrid ASSISTments Mobile Application

An Interactive Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfilment of the requirements for the
Degree of Bachelor of Science

By
Andrew Petit, Ama Biney, Dario Martinovic

Date:
May 2, 2017

Submitted to:

Professor Neil Heffernan and Cristina Heffernan
Worcester Polytechnic Institute

ABSTRACT

The goal of this IQP project was to develop a mobile application that could be used universally across any platform with the capability to connect to the internet. This app will allow problems created on the ASSISTments website to be used by students to quiz themselves on material that could be on their upcoming exams. This report details the development process and talks about the design of the app in full detail.

ACKNOWLEDGEMENTS

First and foremost, we would like to express our deepest gratitude to Professor Neil Heffernan and Christina Heffernan for giving us the opportunity to work on this IQP. It was a great personal achievement to be a part of a project that could directly benefit the population of students around us.

We would like to thank Chris Donnelly and David Magrid for their eagerness to help and constant direction with code development.

Thank you to Professor Dittami and Professor Cooper-Morgan for allowing us to have insight to their courses. Without their generosity, we would not have been able to gather as much information about chemistry on the WPI campus as we did.

We would like to thank Emily Hao for working diligently with us to create proper logos and user interface.

Lastly, we would like to thank Yihao Zhou in his efforts in development and with researching HTML5 and other frameworks.

TABLE OF CONTENTS

INTRODUCTION

ASSISTments

Over 12 years ago, Cristina and Neil Heffernan created the groundbreaking website named ASSISTments in the Computer Science department at Worcester Polytechnic Institute. As it stands now, ASSISTments is an online system designed for teachers to test their students' performance on certain questions. This platform gives teachers data on where their students struggle the most, while simultaneously giving students immediate feedback on what areas they should work to improve.

Today, ASSISTments is used by over 500 K-12 teachers across the country. The mission is quite simple: to improve education through scientific research while not compromising student learning time. The service, moreover, aims to continuously improve the processes of teaching and learning by incorporating validated educational programs and applied technology in classrooms everywhere, as such ASSISTments is offered free of charge so that it is accessible to everyone. To fund ASSISTments continued development and maintenance, the service relies on grants and other contributions.

As mentioned, the program works by providing teachers with detailed feedback on their students' progress with certain class content. With this feedback, the teacher can determine which areas their students' struggle with the most, and can focus more class time to these topics. Since the ASSISTments service also corrects problems for

teachers automatically, teachers spend less time correcting and more time reflecting on student work, responding to student questions and planning effective lessons.

QuizASSIST

In QuizASSIST, the objective was to streamline the ASSISTments service into a mobile application that would utilize game elements to make improving skills more enjoyable. Instead of providing them with assigned questions by an instructor, students would be allowed to choose among a number of different course topics based upon an area the student feels that they should focus on. Random questions would then be generated for the student based upon the chosen content. A score report is then provided to show areas that the student could look to improve in.

In the latest version of the application, QuizASSIST also provides a challenge mode. This mode allows users to compete against their friends with content generated by the app. Both users will be informed of which player “won” the challenge, thus providing each of the students with an incentive to learn the material so that they can win. This, therefore, increases the chances of improvement in the given field for both players. A future mode will be added over the next few months that will allow multiple students to synchronously work on problems together, providing each other with hints if users get stuck on a given question. A full version of QuizASSIST can currently be found at http://astest1.cs.wpi.edu:8080/quizassist/quizassist-web-template/index.jsp.

CONTENT

A. How Specialize at WPI?

The key to being successful with QuizASSIST is understanding how to specialize the content that will be learned, practiced, and challenged in each specific market. When thinking about how this application would thrive on the campus of WPI, it was important to reflect on what would be a common subject that many students needed as part of the curriculum and would like to practice frequently. At WPI, all majors require some type of basic science regardless of their field of study. According to U.S. News and World Report, in academic year 2016-2017, WPI reported an undergraduate enrollment of 4,299 students [1]. This, combined with the fact that over 400 students took an introductory chemistry course, CH 1010, in A term of 2016, demonstrates that close to 10% of the undergraduate student body took chemistry. Therefore, the CH1010 chemistry content would be the most relevant place to start when building the content for the first version of the QuizASSIST application.

B. Content Retrieval Process

The first step in the initial research was getting a true understanding of what a typical chemistry course consists of at WPI. The book that content is taught from is “Chemistry: An Atoms-Focused Approach” by Thomas R. Gilbert, Rein V. Kirss, and Natalie Foster. Even though the book has just over 20 chapters the introductory chemistry course at WPI, CH 1010, only focuses on the first six chapters. The content of these chapters is broken down into the following subject areas:

Chapter 1- Matter and Energy

- States of Matter
- Forms of Energy
- Classes of Matter
- Properties of Matter
- Atomic Theory-Scientific Method
- Molecular View-Molecular Formula
- COAST: solving problems (collect, organize, analyze, solve, think)
- Making measurements- precision & accuracy, sig figs
- Unit conversions
- Temperature scales

Chapter 2- Atoms, Ions, and Molecules

- Rutherford Model of Atomic Structure
- Nuclides and Their Symbols
- Navigating the Periodic Table
- The Masses of Atoms, Ions, and Molecules
- Moles and Molar Masses
- Making Elements
- Artificial Nuclides

Chapter 3- Atomic Structure

- Waves of light
- Atomic Spectra
- Particles of Light: Quantum Theory
- The hydrogen spectrum and the Bohr Model
- Electrons as waves
- Quantum Numbers
- The sizes and shapes of atomic orbitals
- The periodic table and filling orbitals- electron configuration
- Electron configuration of ions
- The sizes of atoms and ions
- Ionize energies
- Electron affinities

Chapter 4- Chemical Bonding

- Chemical bonds- ionic, covalent, metallic
- Naming compounds and writing formulas- binary ionic compounds of main group elements, binary ionic compounds of transition metals, polyatomic ions, binary molecular compounds, binary acids
- Lewis symbols and lewis structures
- Electronegativity, unequal sharing, and polar bonds
- Vibrating bonds and the greenhouse effect
- Resonance

- Formal charge: choosing among lewis structures
- Exceptions to the octet rule
- The lengths and strengths of covalent bonds

Chapter 5: Bonding Theories

- Molecular shape
- Valence-shell electron-pair repulsion theory
- Polar bonds and polar molecules
- Valence bond theory and hybrid orbitals
- Molecules with multiple “central” atoms
- Chirality and molecular recognition
- Molecular orbital theory

Chapter 6: Intermolecular Forces

- London Dispersion Forces
- Interactions involving polar molecules
- Trends in solubility
- Phase Diagrams: Intermolecular forces at work
- Some remarkable properties of water

Before creating the content about these subjects, it was crucial to get the support from professors who taught this course. This would allow the group to have a strong

understanding on how students were expected to apply their knowledge and how they would be tested. The content group reached out to all the professors who would be teaching this course in the 2016-2017 school year. Many of these professors showed overwhelming support by sending their syllabi and even adding the content group members as an observer to their course website, while others kindly declined. Each section of this course had weekly labs and homework assignments. Depending on the professor, students either took a midterm and a final or had a series of 4 exams during a 7 week long term. After analyzing what was being taught and assigned in this course, it had become obvious that the QuizASSIST application could change the way students review notes and prepare for examination. At this point the goal was to create an application, as an alternate form of studying, to help students perform better on homework assignments, labs, and exams.

The next step was to accumulate as much raw content as possible so that students could test the app once the Graphical User Interface (GUI) was made available. Eventually, with the help of ASSISTments, previous work from professors/students, and newly created content from the featured chemistry textbook, the content for chemistry course, CH 1010, was created. Because of this it was possible to make the first version of the app to be tested among students.

In terms of ASSISTments, the process of gathering and building content took two routes. First, a school in ASSISTments was created and labeled QuizASSIST where all the content for any course or subject will reside. For example, for CH1010, a Chemistry 1010 folder is created which contains the course content divided by the chapters. The

first route was to use the content that was previously in the archives of ASSISTments. For each section or skill builder, a copy of the content was made to create a problem set and then added into QuizASSIST school. In order to create a copy of the content, the Builder Tool inside of ASSISTments is used as shown in Figure 1 below.

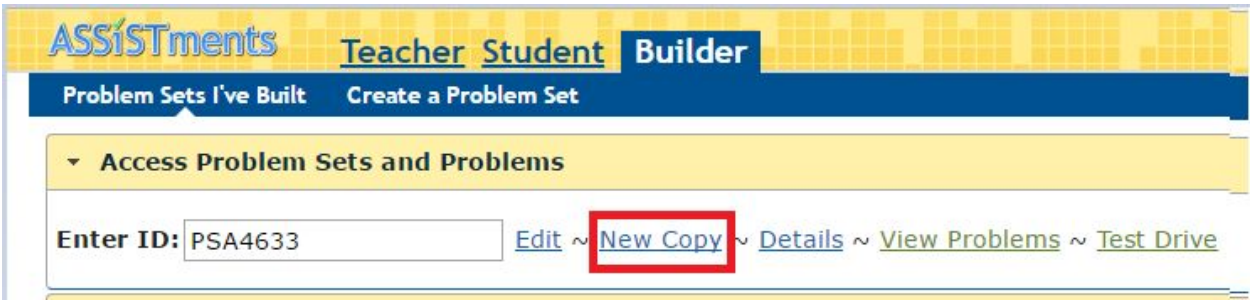


Figure 1 : Copy Problem Set in ASSISTMENTS

By using the approach from Figure 1, it is possible to use the existing problem sets, certified by ASSISTments, to create copies of such problems. This is accomplished by copying the problem set ID that one wishes to reproduce and hitting the “New Copy” option. Therefore, these new copies can be edited to suit the taught content without disturbing the original problem sets. This is a very useful feature because it allows the re-use of content as well as decreasing the time need to create additional content.

The other approach is to build a new problem set, which is a feature made possible through ASSISTments. The old exams and quizzes that were gathered from students were used to create new problems sets. Figure 2 below shows how to build a new problem set.

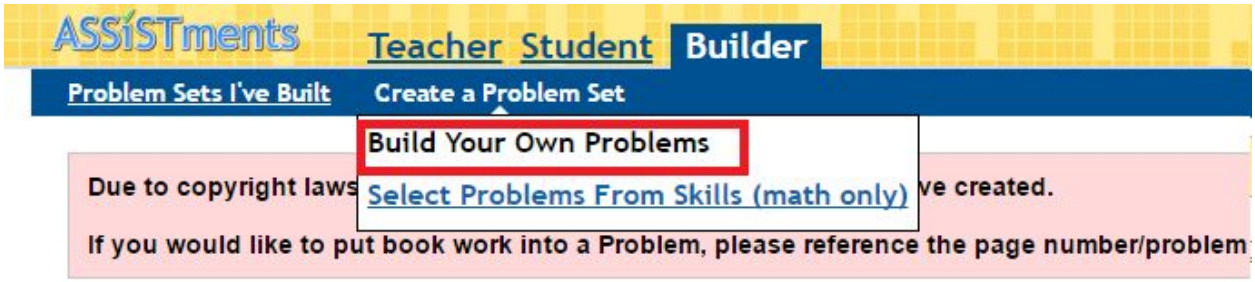


Figure 2 : Build a New Problem Set in ASSISTMENTS

All the problem sets will show up in “Problem Sets I’ve Built” section of the ASSISTments. In order to make the content available to the QuizASSIST application the problem sets must be added to the QuizASSIST school to the appropriate chapters. For example, in order to add a problem set to Chapter 2 in Chemistry 1010, one would have to navigate to “Teacher” option in ASSISTments and click on “Shared in the school: QuizASSIST -> Chemistry 1010 -> Chapter 2 -> Add” and add the appropriate problem set via problem set ID. This is shown in Figure 3 below.

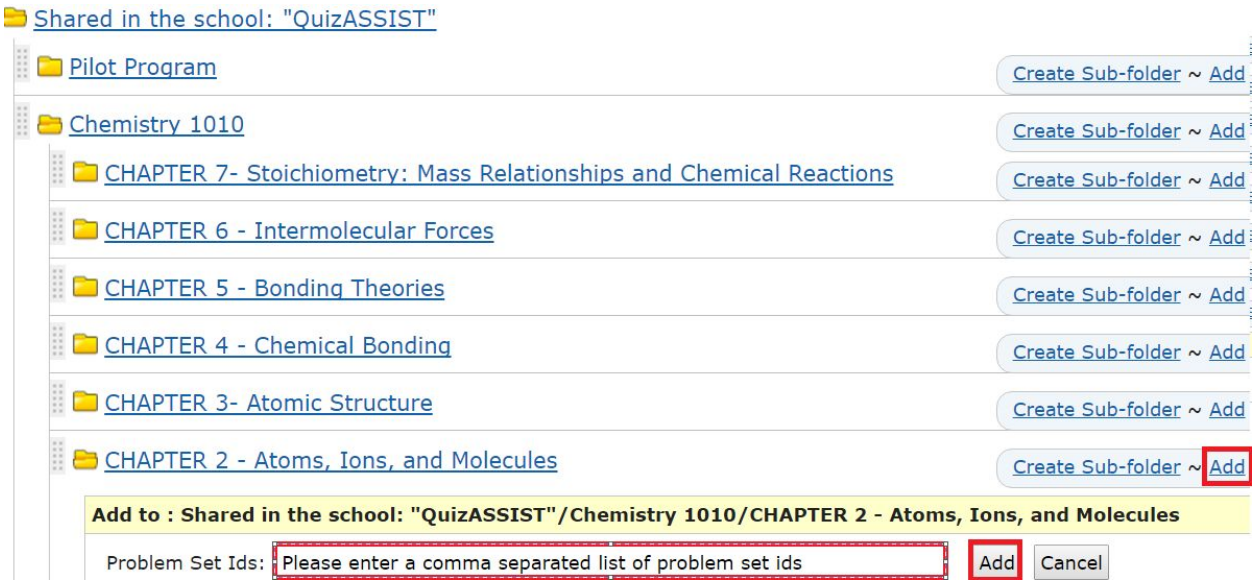


Figure 3 : Add a Problem Set in ASSISTMENTS

Finally, the content for the QuizASSIST application is formed and the folder hierarchy is defined as shown in Figure 4 below.

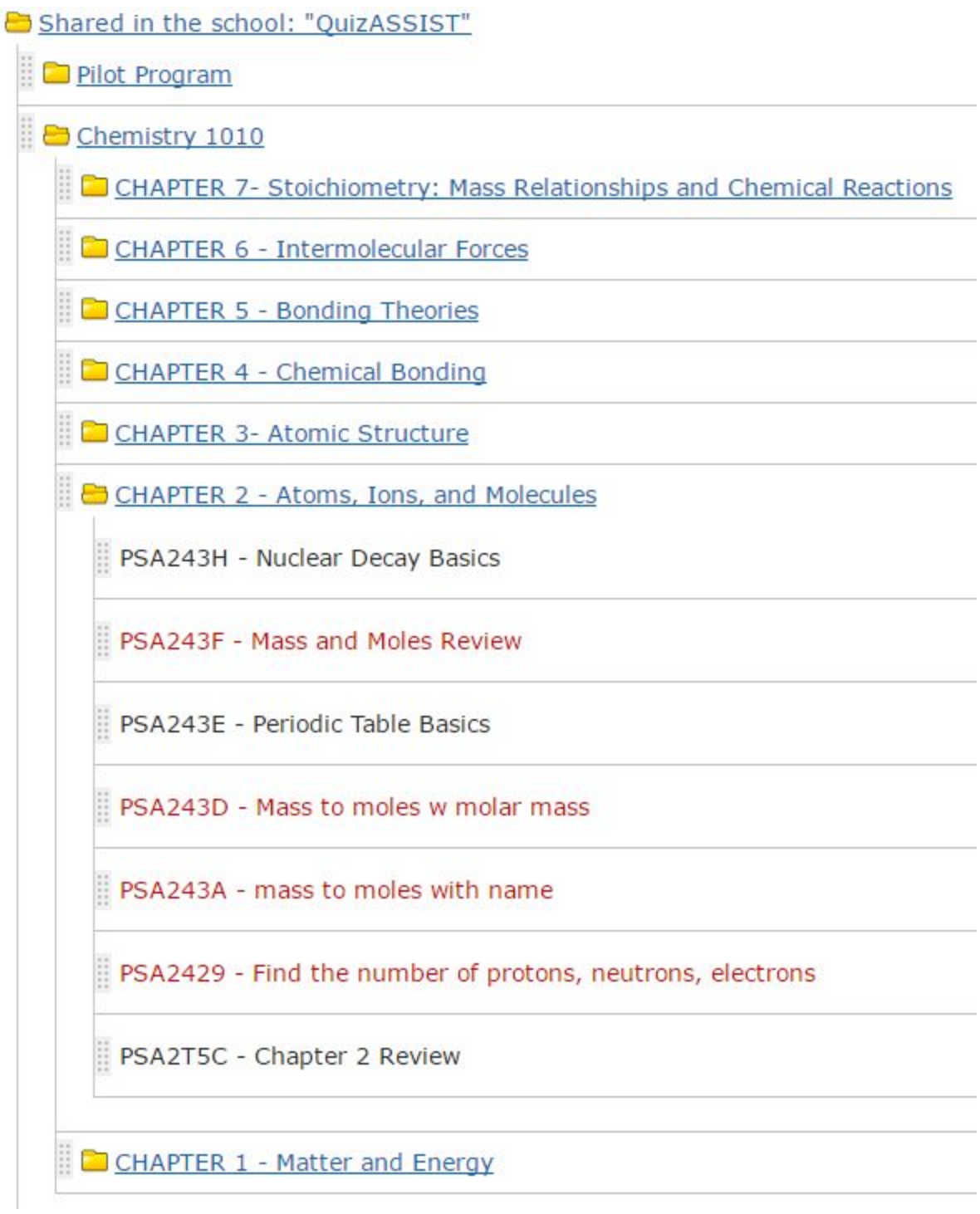


Figure 4 : Content Format on ASSISTMENTS

From Figure 4, it is shown that content is divided based on chapter. For example, inside of the Chemistry 1010 folder, there are folders for each chapter that pertain to this course. Furthermore, inside of each chapter there is a number of different problem sets. The content group had to make sure to create the problem sets, and to ensure that each problem set included questions that are were covered and taught in the CH 1010 course. This involved referring to the syllabi as well as lecture notes, which were provided by chemistry professors. Furthermore, the content was modified and updated as more feedback was received by the chemistry students that chose to use the QuizASSIST application. Finally, each chapter ended up having a number of problem sets from which a user/student is able to choose from. Also notice that each chapter contains a chapter review problem set which is a mix of all the different topics infused with the exam-style questions. This strategy gives the students a choice, to either focus on a specific topic or have a complete chapter review when preparing for exams.

The next step was to choose how students practice the content when in QuizASSIST app. It was decided that three types of questions would be made available: “multiple-choice”, “check-all-that-apply”, and “fill-in the blank”. This was done in order to provide more flexibility and a better user experience. Luckily, ASSISTments already offers options to pick between these question types when content is created.

For example, if the student is asked to perform a very specific calculation the result would have to be typed. This is a good example for a “fill-in-the-blank” question. Figure 5 below shows an example of such a question for CH1010 content.

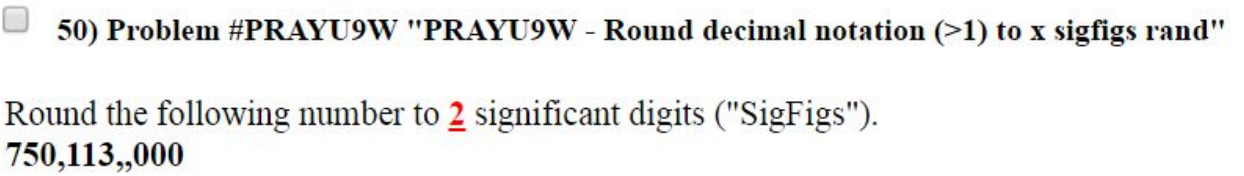


Figure 5: An example of a "fill-in the blank" question

On the other hand, some problems are better suited for as a multiple-choice type question. For example, if there is only one correct answer, multiple-choice is best used to check whether students have learned the specified content. Figure 6 demonstrates an example of a multiple choice question for the CH1010 course.

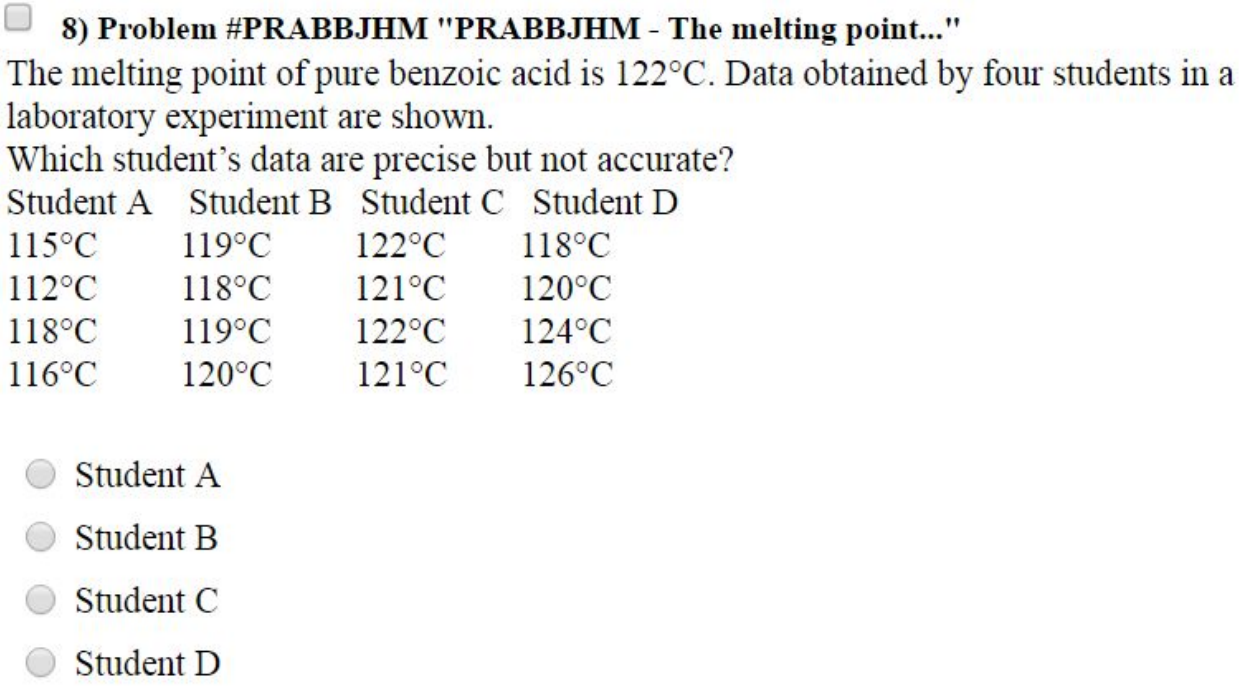


Figure 6: An example of "multiple-choice" question

Lastly, when a question has a number of correct answers “check-all-that-apply” is used. This is best illustrated with an example of a question from the CH1010 content, as shown in Figure 7 below.

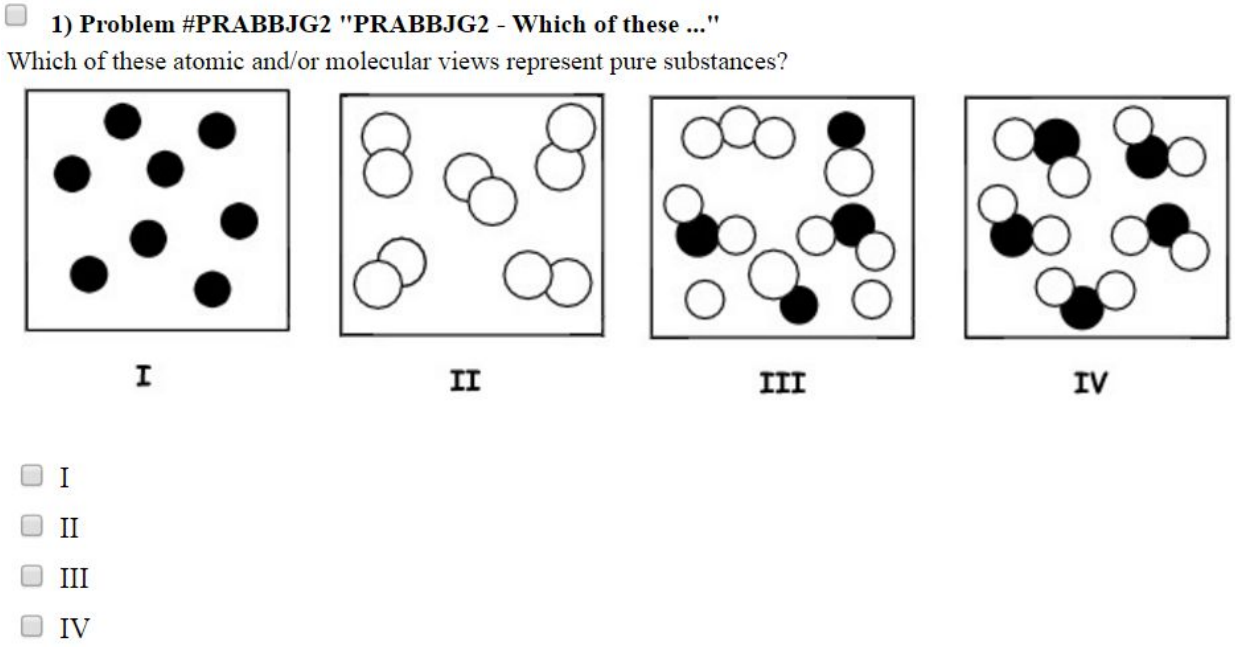


Figure 7: An example of “check-all-that-apply” question

IMPLEMENTATION

METHODOLOGY

Development Platform

The goal of the IQP was initially to develop QuizASSIST so that it would be available for both Android and iOS operating systems. Initially, the thought was that it would be appropriate to focus on making the application work with one of these two platforms before beginning to move over to the other, but this proved impossible when

development began to start for Android, as the design of the iOS build would not have been easy to replicate in Android. To account for this issue, during C-term the development team worked on migrating the application, which at the time had been entirely constructed in Objective-C, over to a hybrid-mobile application. In this implementation, a web page is created that is then coded to open in an iOS or Android application. As such, a consistent design would be created for both operating systems. This would also allow users to use the application on their laptop, desktop, or other internet compatible device as the web page could be accessed by simply going to the URL. In turn, this would help provide the team with more feedback as the application would be accessible to more people.

The app, as mentioned, was initially coded using Objective-C, but, with the C-term migration, the development team moved all existing client logic over to javascript. The new universal design was then created using JSP and CSS styling. The team especially relied on the HTML5 and bootstrap frameworks to make the design look fresh.

On the server side, Java is used to access the database and process HTTP requests sent from the client. Most of the server functionality, relies directly on the Spring Framework, and software development kit provided by ASSISTments.

Design Features

1. Universal Design

To make the application function on any internet-capable device, the original iOS based application was migrated over to a combination of javascript, HTML/JSP, and CSS. While an iOS application still exists, it merely opens to the index page URL of the new web based application. In addition, this URL:

http://astest1.cs.wpi.edu:8080/quizassist/quizassist-web-template/index.jsp

can be copied and pasted into all compatible internet browsers to access QuizASSIST. This includes all mobile browsers that are found on Android and iOS based devices and most computer browsers such as Chrome, Safari, or Microsoft Edge. Therefore, a user no longer needs to have downloaded QuizASSIST from the App Store to have access to the application.

This change to a universal design was done in the hope that the application would become more accessible to a larger audience. Since not all people own a iPhone or iPod touch, making the software available on all devices seemed like a good way of increasing the number of users. While this idea has yet to be tested due to the application being unavailable in d-term due to the migration, a future IQP group would be able to determine the impact of this move on increasing the number of clients.

2. User Account

The previous IQP worked to separate user accounts from ASSISTments because there was a concern that some potential QuizASSIST users might not know about

ASSISTments. As such, the users would be prompted to visit the ASSISTments website to create an account before they would be allowed on the application, which could have proven to be an inconvenience.

However, a recent update to the ASSISTments provided software development kit addressed this concern by creating a user login page. While this login page does offer a user the option to login with an ASSISTments account, it does not limit them to using this as a means of logging into QuizASSIST. As there are now options to login with a Facebook account or Gmail account. Furthermore, this page gives the user the option of creating an ASSISTments account on the spot so no further redirection needs to happen.

To implement this new change, the development team simply redirects a user who wants to visit the homepage of QuizASSIST to the login portal. Once the user logs in, the user is then directed back to the homepage of QuizASSIST. In addition, a cookie is stored in the user's browser that will allow the user to return to the homepage of QuizASSIST without needing to log back in again on subsequent visits unless the user deletes the cookie.

3. Content

All content provided by QuizASSIST is currently taken directly from the ASSISTments webpage. The high majority of this content will be teacher made, where teachers will be able to visit ASSISTments and post questions for any number of course topics. This content will be subsequently pulled by QuizASSIST, and then randomly assigned to a student after the student chooses which content they wish to work with in

the application. Other sample content is provided on ASSISTments and will be made available to students using the application in the event that teachers do not wish to post their own content for a certain course. As of right now, only Chemistry and Arithmetics are offered, but the team anticipates the amount expanding when QuizASSIST gets adopted by more classrooms. A more detailed analysis of provided content can be found starting on page 7.

4. Problems

The application was designed to utilize multiple choice, fill in, and check all that apply type questions to test the user on content that is pulled from the ASSISTments webpage. With ASSISTments, teachers are able to create or post any of these three type of questions, and this content will then be made available for use on QuizASSIST. In addition to teacher made content pertaining to specific class topics, students will be able to choose from a variety of sample problems which will give them the opportunity to test themselves in various different subject areas.

While the current implementation does not provide for hints, a future study group mode will offer the ability for students to ask their peers for help with certain questions they may be having a difficult time answering. This mode will give students the ability to work with other students in their class on challenge topics without having to be in the same room, allowing students to help one another master the content together.This differs from the current challenge mode, where students are solely competing against one another on their knowledge of the chosen content. Hints will become available in both challenge and practice modes in future iterations, but these hints will be taken from

the ASSISTments webpage, and students will receive a slight deduction in the number of points that they can earn for answering the question correctly so that these modes retain their competitive element.

PREVIOUS IMPLEMENTATION

Initial Plan

QuizASSIST began as an iOS only application with a client that was developed exclusively in Objective-C using the Xcode 8 IDE. The initial plan was to create a fully functioning application that would be available in the Apple App Store before expanding QuizASSIST to work with different platforms such as Android and Windows Mobile OS. The intention of doing this was to garner feedback about what QuizASSIST did well, what should be improved in the application, and what should be added as future functionality to streamline the expansion process. While the application was never approved for distribution on the Apple App Store, the team was able to successfully distribute QuizASSIST via Apple's external testing platform, TestFlight. This distribution software allowed the team to provide a copy of QuizASSIST to an entire CH1010 class at Worcester Polytechnic Institute to gather user feedback. However, after gathering feedback from users of this class, a decision was reached in mid-February 2017 to make the application work universally on any device capable of connecting to the Internet to increase accessibility to the application.

Icons, Logos, Color-Scheme, and Splash Screen

The last version of the application featured a theme that made the application appear to be more aimed at elementary and middle school students. This has since been corrected with the recent version of the application, to make the intended audience more universal. Nevertheless, pictured below are images taken from the old application.

Figure 8: The old icon for QuizASSIST.

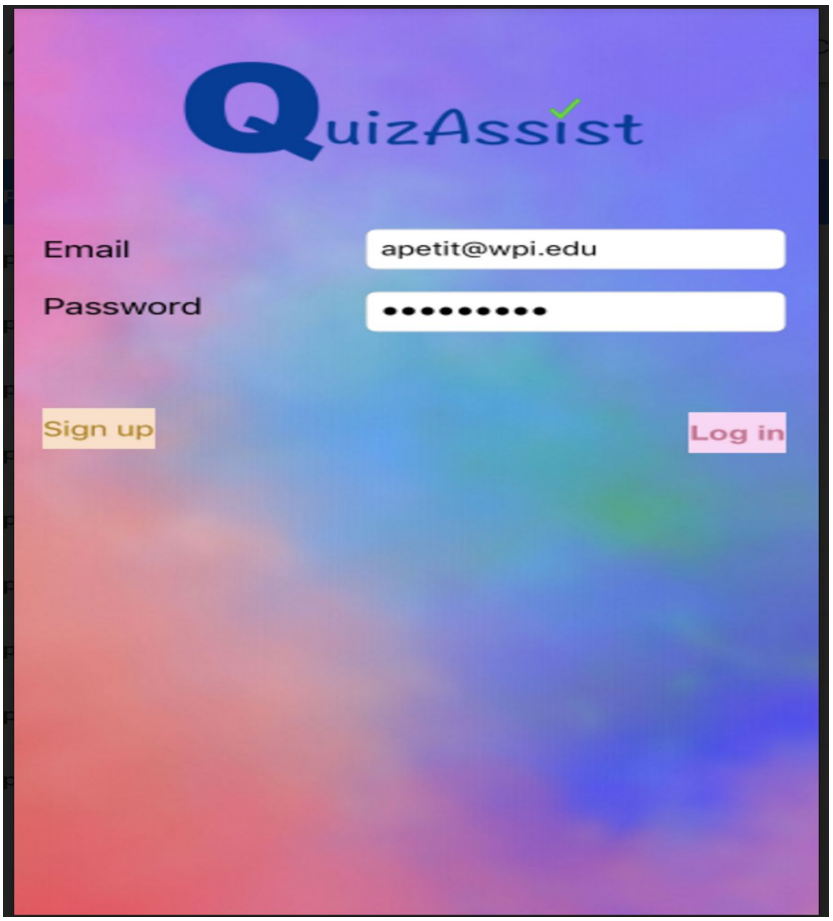


Figure 9: The original login page for QuizASSIST.

Figure 10: The fireworks were considered slightly distracting from the original recap page from the initial application.

Initial Application Flowchart

Featured below is the flowchart of the initial iOS application. The main differences between this one and the one featured on page 32 exist with color scheme, design, and login/signup screens. In regards to these page changes, the newest version of the application relies on the SDK provided login portal to create and log users in. This allows the newest version of QuizASSIST to simply redirect users to this login portal URL instead of having to provide its own login page for users. Other than these few changes, the model has remained relatively consistent.

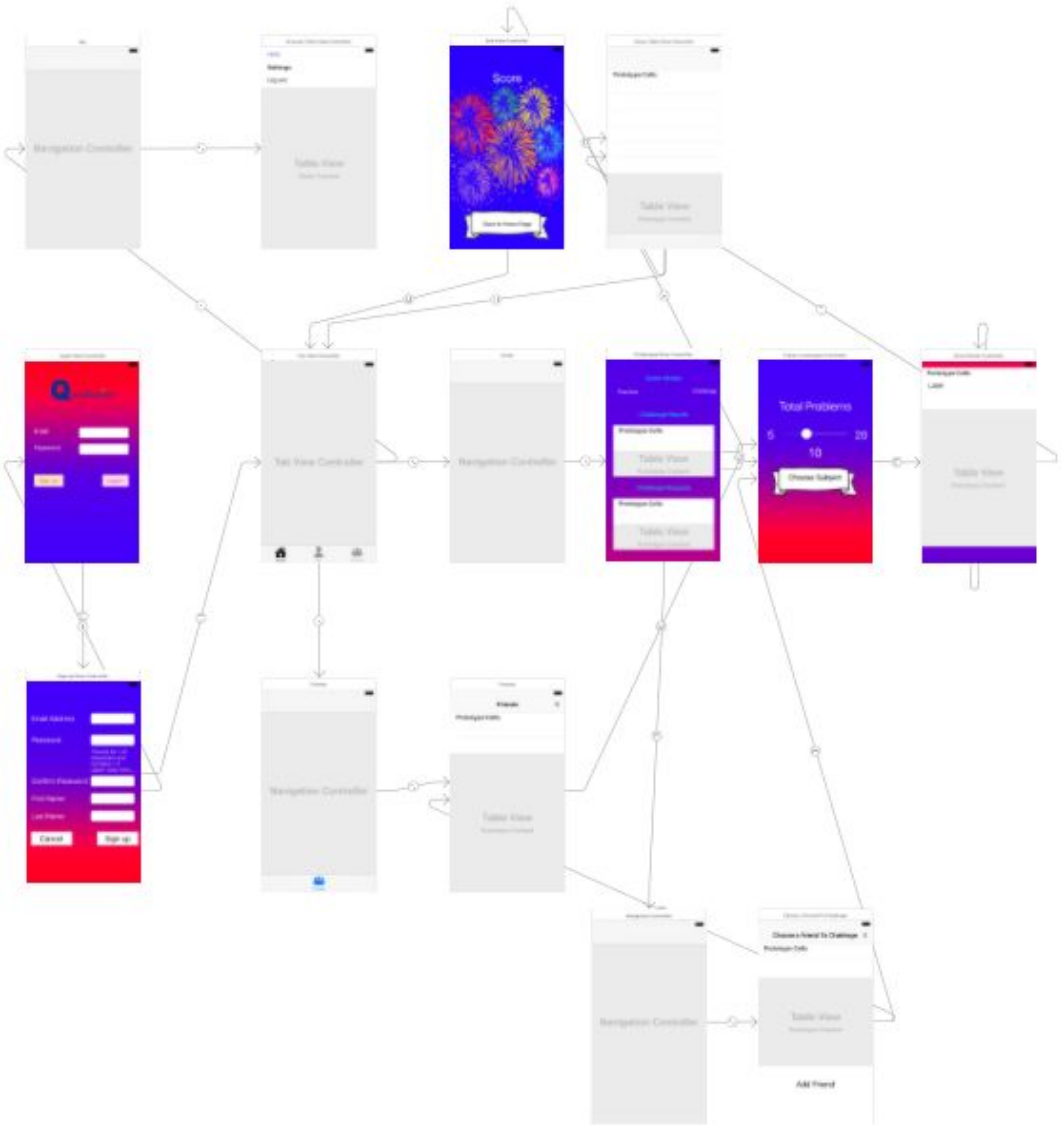


Figure 11: The old program flowchart

CURRENT IMPLEMENTATION

Icons, Logos, Color-Scheme, and Splash Screen

The Icon, Logo, Color Scheme, and Splash Screen for the application were all developed by the team's UI engineer, Emily Hao. Over the past several terms, the

application has undergone multiple revisions to all images and colors, but finally settled on a clean and simple design that appears less childish than previous designs in an effort to make the application appear more suitable for people of all ages. With this current design, the application uses three colors - pink, blue, and green - as the main focal point. These colors represent the three game modes that are offered by QuizASSIST; pink for practice mode, blue for challenge mode, and green for study group mode.

This new focus on these three colors is evident even from the initial splash screen of the iOS application. Gone is the old logo where a face appeared inside of the Q, and in its place are three vertical bars near the word QuizASSIST with each of the bars being one of the three aforementioned colors.

Figure 12: The splash screen that appears when first opening the iOS version of the application.

The following images highlight this new theme throughout the application itself:

Figure 13: Buttons featured on the index page of QuizASSIST used to pick a game mode.

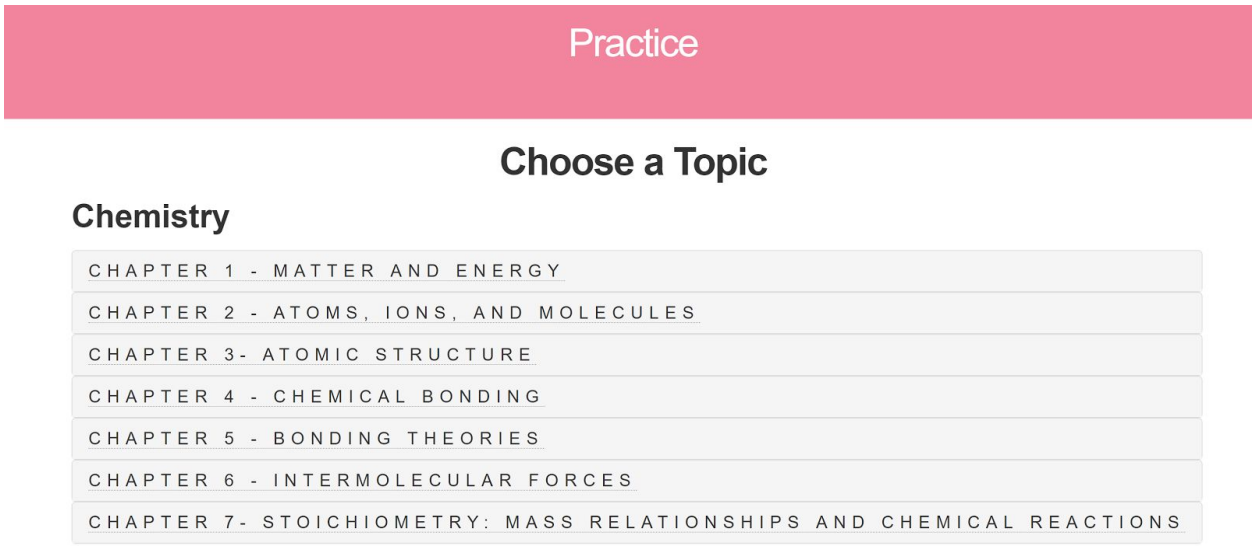


Figure 14: The appearance of the choose topic page when the user is currently in practice mode.

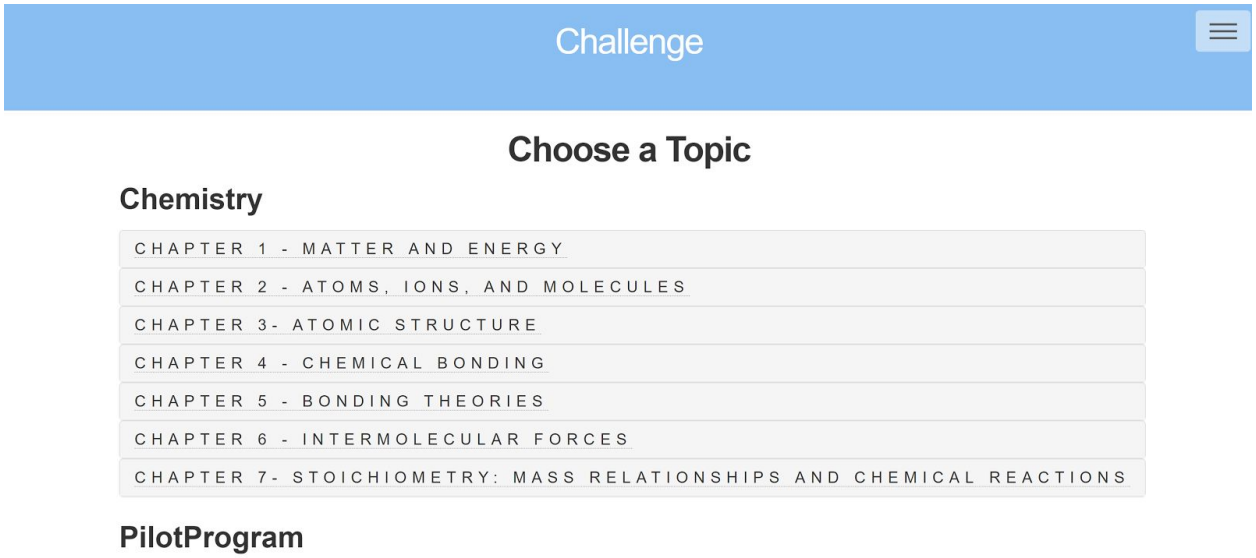


Figure 15: The appearance of the choose topic page when the user is currently in challenge mode.

Database Model

Below is a model of the database used by QuizASSIST. As it currently stands, there has been no need for the usage of a foreign key in any tables, therefore, no relationships exist between any of the tables. Nevertheless, the model is given to show the different data that is stored in each of the used tables. The USER_RESPONSE table is used for logging a user's answer to a given question and records what the actual answer to the question was. The USER_FRIENDS table is used to log each of a given user's friends. The CHALLENGE_SET table is used to log all of the problem IDs from a given challenge quiz. These problem IDs will be sent over to the user(s) that the challenger wanted to challenge to display an identical quiz to these people. The CHALLENGE_REQUEST table is currently unused, but, in the future, will be used to allow any number of problems to be used in a challenge instead of the current limit of twenty questions. Both the USER_REFERENCES and MOBILE_DEVICES tables were

used in the old version of the application, but are no longer used. These tables will be deleted by the next IQP group working on this application.

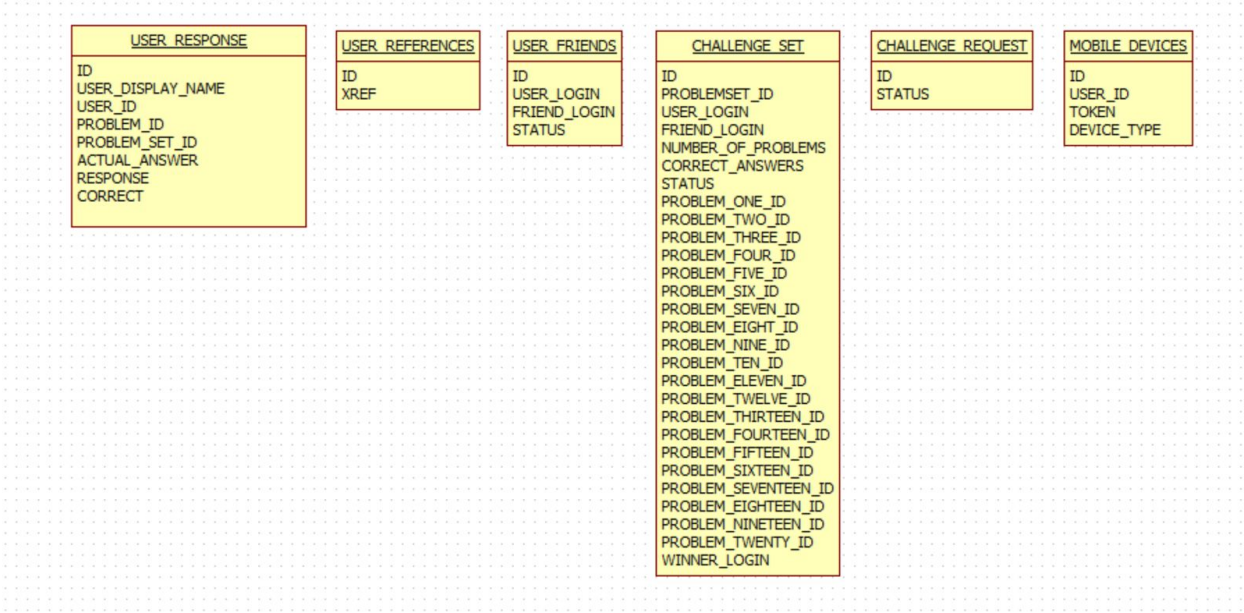


Figure 16: The database diagram for the database used by QuizASSIST.

Application Flowchart

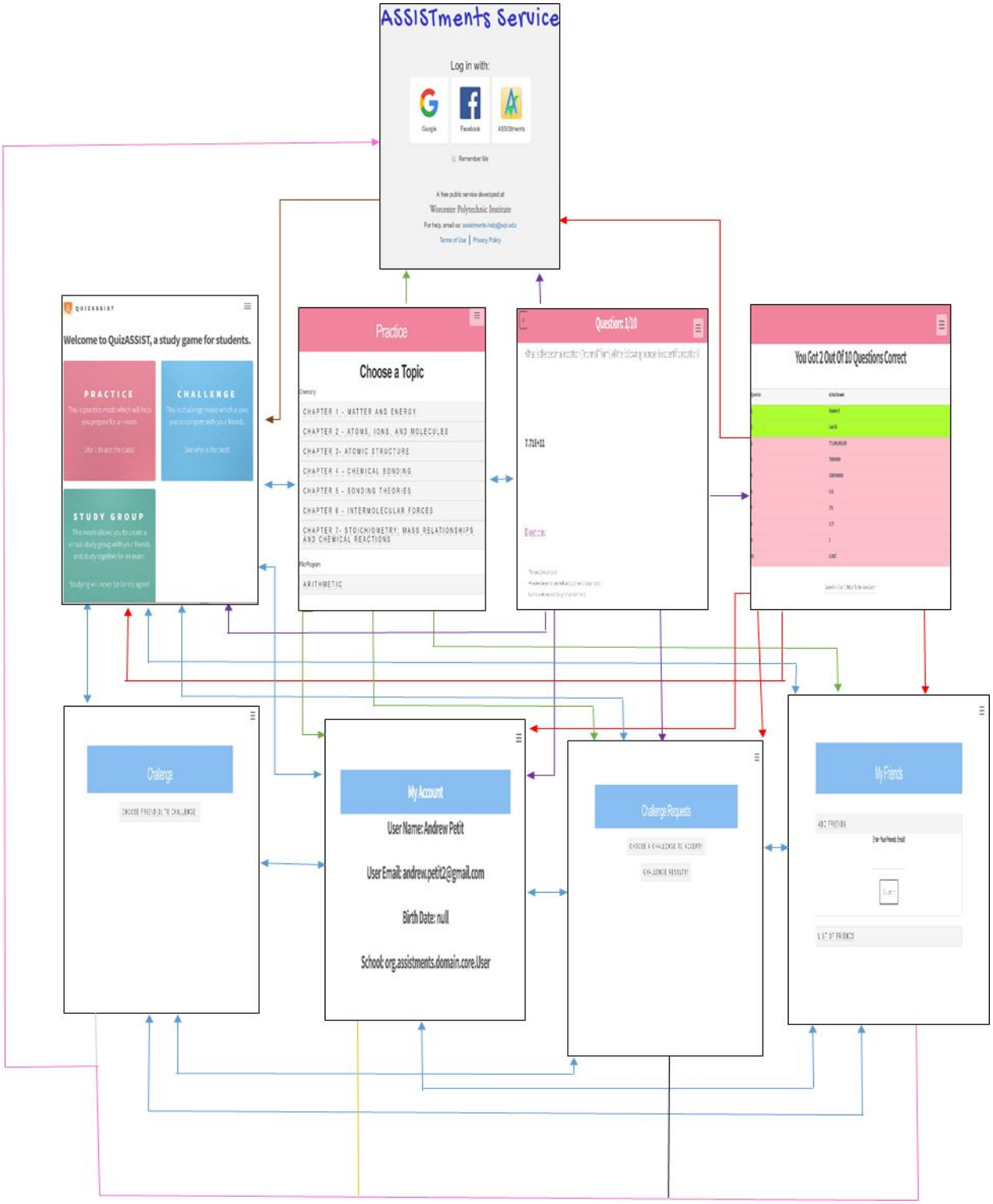


Figure 17: Current Application Flow Chart

The above chart shows how the application currently operates. Each of the boxes shown represents a current JSP page in the application. Arrows represent that navigation is possible to the page the arrow is pointing at from the given page. Each of the elements will be discussed in detail in the following sections.

1. Login Portal

When a user first visits the index page of the hybrid mobile application, they will be directed to the sdk provided login portal. This is true regardless of whichever way the user accessed the application - whether it be through the iOS application or by visiting the application URL in a browser. Once at the login portal, the user can access QuizASSIST by entering their google login information, facebook login information, or ASSISTments login information. If the user does not have any of these credentials, he or she may sign up for an ASSISTments account directly on the login portal page. Once the user has been authenticated, they will be redirected to the QuizASSIST index.jsp page which will connect the user with all other functionality of the application.To prevent users from having to login every time they request the index.jsp page, a cookie is stored on the user's browser, letting the service know that they had been authenticated before. Only after the cookie is deleted either manually or by logging off will the user be prompted to login to QuizASSIST again.

Featured below is a copy of sample javascript code taken directly from the loginViewController.js file. This code shows how the user is redirected to the login page if the cookie which proves if the user was authenticated is not present. In this case, the user is redirected to the login portal website. Otherwise, QuizASSIST determines that

the user does not need to login again. In this case, an ajax request is called which will return the user's login information.

```
//if user is not logged in on the computer - log him/her in
var myCookie = getCookie("ASSISTmentsService");
if (myCookie == null){
    console.log('astest1.cs.wpi.edu:8080/loginPortal?callingPartnerRef=mobileAppService&callbackURL=' + loginURL);
    window.location.href = 'http://astest1.cs.wpi.edu:8080/loginPortal?callingPartnerRef=mobileAppService&callbackURL=' + loginURL;
} else { //user had previously logged
    console.log(getLoginURL);
    var loginInfo = $.ajax({
        type: "GET",
        url : getLoginURL,
        success : function(data){
            console.log("success");
            console.log(data);
            localStorage.setItem("displayName", data.displayName); //just in case the user cleared local storage, make sure we reset
            localStorage.setItem("userId", data.databaseId);
            localStorage.setItem("userLogin", data.loginName);
            localStorage.setItem("birthDate", data.birthDate);
            localStorage.setItem("school", data.school);
            },
            error : function() {
                console.log("fail");
                }
            });
    }
```

Figure 18: Sample code from loginViewController.js that determines whether or not a user needs to log into QuizASSIST.

2. QuizASSIST Index Page

The QuizASSIST index page serves as the application's home page. On this page, a user can select the menu icon in the upper right hand corner to direct themselves to their account information, their friend page, their pending/competed challenge page, and to also log out of the application. The user will also use this page to create a practice exam for themselves or initiate a challenge against another user by selecting either the practice or challenge button respectively. While a Study Group button is also shown on the index page, this mode is currently unavailable, but will be made available sometime over the coming months.

Featured below is HTML code taken from the index.jsp page to show how QuizASSIST uses both of the HTML5 and Bootstrap frameworks to maintain the unique design of the home page. In each of the HTML elements of the below code, classes or ids are assigned to the elements. The HTML5 and Bootstrap stylesheet pages then

modify their design based upon this identifier. The below code shows how this is done for the practice, challenge, and study group buttons on the index.jsp page.

```
<!-- Main -->
    <div id="main">
        <div class="inner">
            <header>
                <h1>Welcome to QuizASSIST, a study game for students.</h1>
            </header>
            <section class="tiles">
                <article class="style1">
                    <span class="image">
                        <img src="${pageContext.request.contextPath}/resources/images/pic01.jpg" alt="" />
                    </span>
                    <a href="questions.jsp">
                        <h2>Practice</h2>
                        <div class="content">
                            <p>This is practice mode which will help you prepare for an exam.</p>
                            <P>Use it to ace the class!</P>
                        </div>
                    </a>
                </article>
                <article class="style2">
                    <span class="image">
                        <img src="${pageContext.request.contextPath}/resources/images/pic02.jpg" alt="" />
                    </span>
                    <a href="challenge.jsp">
                        <h2>Challenge</h2>
                        <div class="content">
                            <p>This is challenge mode which allows you to compete with your friends.</p>
                            <p>See who is the best!</p>
                        </div>
                    </a>
                </article>
                <article class="style3">
                    <span class="image">
                        <img src="${pageContext.request.contextPath}/resources/images/pic03.jpg" alt="" />
                    </span>
                    <a href="studygroup.jsp">
                        <h2>Study Group</h2>
                        <div class="content">
                            <p>This mode allows you to create a virtual study group with your friends and study together for an exam.</p>
                            <p>Studying will never be lonely again!</p>
                        </div>
                    </a>
                </article>
            </section>
        </div>
    </div>
```

Figure 19:Sample HTML code taken from index.jsp that shows.

3. Select Topic Page

The select topic page is where a user can select what content they want to test themselves or challenge others with. This page can be accessed in one of two different ways. From the index.jsp page, the user can select the practice box which will take him or her directly to this page, or the user can select the challenge box on that page which, after the user selects a friend or friends to challenge, will be redirected to the select topic page.

The select topic page will appear differently depending upon the means by which the user originally accessed the page. If the user were to get to this page by selecting

the practice button on the index page, the select topic page will appear pink and feature the header “Practice.” If the user gets to this page by selecting challenge mode and subsequently a friend to challenge, the select topic page will appear blue and feature the header “Challenge.” The images below highlight the differences between the two designs.

Figure 20: The images above show how the choose topic page will appear based upon the different means of accessing the page.

In terms of code, the select topic page relies on the unionFolderViewController.js file to build the drop down lists that display the content taken from the ASSISTments webpage. Essentially, the file features code that parses all the content retrieved by a REST based service, and then uses the “innerHTML” function to show each specific topic as a row in the drop down list HTML elements. Provided below is a sample of code from unionFolderViewController.js that shows how this is done.

```
function processData(data) {
    var list = document.getElementById('list');
    var chapters = "";
    // write out chemistry
    Object.keys(data).forEach(function(key0) {
        chapters = "<li><p style = \"font-family: inherit; font-size: 30px; font-weight: bold\">" + key0 + "</p>";
        var index = 0;
        //build the drop down list
        for (var k = 0; k < arr.length; k++){
            index++;
            chapters += "<div class=\"panel-group\" id=\"accordion\">"
                + "<div class=\"panel panel-default\">"
                + "<div class=\"panel-heading\">"
                + "<h4 class=\"panel-title\">"
                + "<a data-toggle=\"collapse\" data-parent=\"#accordion\" href=\"#collapse"
                + key0 + index + "\">"
                + arr[k]
                + "</a>"
                + "</h4"
                + "</div>"
                + "<div id=\"collapse"
                + key0
                + index
                + "\" class=\"panel-collapse collapse\">"
                + "<ul class=\"list-group\">";
            Object.keys(data[key0][arr[k]]).forEach(function(key2) {
                // console.log(key2);
                chapters += "<li class=\"list-group-item\" type=\"button\" class=\"btn btn-primary\" onclick=\"processProblemset("
                    + data[key0][arr[k]][key2].Id + ")\">" + key2 + "</li>";
            })
            chapters += "</ul></div></div></div>";
        }
        chapters += "</li>";
        list.innerHTML += chapters;
    });
```

Figure 21: Sample code from unionFolderViewController.js. Highlights how the application creates a drop down list to fit the content taken from ASSISTments.

4. Game Page

The game page is where QuizASSIST actually provides the user with questions to answer. In the current implementation, QuizASSIST provides 10 random problems to the user based upon the topic they chose to work with when on the select topic page. The first of these questions is immediately shown to the user upon their selection of a topic, but, in the event that the user chose a topic which they do not wish to work on, the user is allowed to return to the select topic page at any time by pressing the “X” icon in the upper left hand corner of the game page. To access subsequent questions in the set, the user must answer the multiple choice, check-all, or fill-in question which they are currently shown. After doing so, they will be prompted to press the “Next Question” button which will take them to the next question in the set. If, however, there are no

further problems in the set, the user will be taken to the recap page to view their performance on the ten questions.

The game page was designed to provide immediate feedback to the user as well as feedback to teachers, researchers, and other professionals that could benefit from knowing how students do on certain questions. To do this, QuizASSIST will let users know if they got a question right, or, in the case that the question was solved incorrectly, provide the user with an answer to the problem so that they can best learn from their mistake. In addition, QuizASSIST automatically logs a user's response to a question in the USER_RESPONSE table of the database. This makes it easy for teachers and other professionals to access the information, allowing these people to come up with strategies to best improve student performance.

To log user responses, QuizASSIST uses the Javascript framework, JQuery - namely the AJAX functionality of JQuery - to make a POST request to the .../logging/userResponse service supplied by the QuizASSIST server. The server then updates the database with the user display name, the user response, and the actual response. A more comprehensive discussion of the Javascript, HTML, and CSS code used to create the game page is provided in Appendix C.

5. Recap Page

A user is taken to the recap page upon successful completion of a practice exam or challenge. This page provides a recap of the exam the user just took, letting the person know which problems they got right and incorrect. If the user was challenging another player when this page is accessed, quiz information is recorded in the database

so that the user's friend will be notified that they have a pending challenge to complete. To leave the page, the user can either select the link at the bottom of the page that will redirect them directly to the home page, or can select the menu icon at the top right of the page and choose to visit any of the pages listed (this includes the home page, the challenge results page, the friends page, and the account page).

To show user responses, QuizASSIST adds every answer the user makes into an array that is subsequently stored in the website's local storage, essentially an enhanced way of storing cookies. This array is then retrieved on the recap page, and is parsed to determine which problems the user answered correctly and which ones they answered incorrectly. After this is done, the results are then showed to the user in the table on the recap page. The sample from the endViewController.js shows how the recap page does this.

```
//get all the necessary elements
var correct = document.getElementById('numCorrect');
var table = document.getElementById('table1');
var correctArray = localStorage.getItem('correctAns');

//parse the array of all correct answers
correctArray = JSON.parse(correctArray);
console.log(correctArray);

//parse the problem set
var problemSet = localStorage.getItem('problemSet');
problemSet = JSON.parse(problemSet);
console.log(problemSet);

//parse the array of all incorrect answers
var incorrectArray = localStorage.getItem('incorrectAns');
incorrectArray = JSON.parse(incorrectArray);
console.log(incorrectArray);

//go through and figure out which answers were correct and get the actual correct answer
for (var i = 0; i < correctArray.length; i++){
    //update the table when we get it done.
    console.log(correctArray[i]);
    console.log(problemSet[correctArray[i]]);
    var rightAnswer = "";
    for (var j = 0; j < problemSet[correctArray[i]].answers.length; j++){
        if (problemSet[correctArray[i]].answers[j].isCorrect == true){
            rightAnswer = problemSet[correctArray[i]].answers[j].answerText;
        }
    }
    console.log(rightAnswer);
    table.innerHTML += "<tr style=\"background-color:greenyellow\"><th>" + correctArray[i] + "</th><th>" + rightAnswer + "</th></tr>";
}

//go through and figure out which answers were incorrect and get the actual correct answer
for (var k = 0; k < incorrectArray.length; k++){
    console.log(incorrectArray[k]);
    console.log(problemSet[incorrectArray[k]]);
    var rightAnswer = "";
    for (var j = 0; j < problemSet[incorrectArray[k]].answers.length; j++){
        if (problemSet[incorrectArray[k]].answers[j].isCorrect == true){
            rightAnswer = problemSet[incorrectArray[k]].answers[j].answerText;
        }
    }
    console.log(rightAnswer);
    table.innerHTML += "<tr style=\"background-color:pink\"><th>" + incorrectArray[k] + "</th><th>" + rightAnswer + "</th></tr>";
}
```

Figure 22: Sample code from endViewController.js. The code shows how the program determines which questions the user got right or wrong.

6. Challenge Page

The challenge page can be accessed by a user upon selecting the challenge button on QuizASSIST's index page. This page only serves as a means to allow the user to select one or more friends to challenge. Once a user selects a friend(s) to challenge, they will be redirected to the choose a topic page to continue the process of sending a challenge out to a user.

7. Account Page

The account page provides users with their login information. This includes both the email that their account is registered under and the user's display name. To access the account page, the user should press the menu icon in the upper right hand corner of any QuizASSIST page and then press the header "My Account. " In future iterations, the application will supply the user with the ability to select an avatar from a collection of images provided by the application to provide for a more personalized user experience.

```
<body>
    <div id="wrapper">
        <header id="header">
            <div class="inner">
                <div style="background-color: #89bef2; color: white; padding: 20px; height: 100px;">
                    <div class = "container4">
                        <center><h1>My Account</h1></center>
                    </div>
                </div>
                </br>
                <div id="infoDiv" style="width:auto; height:auto;">
                    <table id="table" style="float: left; max-width:50%; max-height:100%; height:auto;">
                        <tr id="tableRow">
                            <th>
                                <td id="userName">User Name: </td>
                            </th>
                        </tr>
                        <tr id="tableRow2">
                            <th>
                                <td id="userEmail">Email: </td>
                            </th>
                        <tr>
                    </table>
                    <img id="img" src="${pageContext.request.contextPath}/resources/images/noImage.jpg" style="float: right; min-width: 15%; max-width: 30%; max-height: 30%;">
                </div>
            </div>
            <nav>
                <ul>
                    <li><a href="#menu">Menu</a></li>
                </ul>
            </nav>
        </header>

        <nav id="menu">
            <h2>Menu</h2>
            <ul>
                <li><a href="index.jsp">Home</a></li>
                <li><a href="account.jsp">My Account</a></li>
                <li><a href="friends.jsp">My Friends</a></li>
                <li><a href="challengeRequests.jsp">Challenge Requests</a>
                <li><a onclick="return logOut();">Log Out</a></li> <!--<a href="http://astest1.cs.wpi.edu:8080/loginPortal?callingPartnerRef=mobileAppService&callbackURL=http://localhos
            </ul>
        </nav>
    </div>
```

Figure 23: HTML code used to build the account page.

The account page currently uses a table to store all the user information with a small image of a white figurine directly to the right of the table if the device size screen size is large or above the table if the device screen size is small. The code below shows how the application auto adjusts the format of this layout based upon screen size.

```
<script>
    $(window).resize(function() {
        var infoDiv = document.getElementById('infoDiv');
        var width = $("#infoDiv").width();
        console.log(width);
        var width4 = $("#table").width();
        var width2 = $("#userName").width();
        var width3 = $("#userEmail").width();
        var width5 = $("#img").width();
        var infoDiv = document.getElementById('infoDiv');
        var width = $("#infoDiv").width();
        console.log(width);
        var width4 = $("#table").width();
        var width2 = $("#userName").width();
        var width3 = $("#userEmail").width();
        var width5 = $("#img").width();
        console.log(width2);
        console.log(width3);
        console.log(width4);
        console.log(width5);
        if (width4 == 202){
            infoDiv.innerHTML = "";
            infoDiv.innerHTML += "<img id=\"img\" src=\"${pageContext.request.contextPath}/resources/images/noImage.jpg\" style=\"Display: block; margin: auto; min-width: 15%; max-width:
            var table = "<table id=\"table\" style=\"Display: table; margin: auto; max-width:50%; max-height:100%; height:auto;\">";
            table += "<tr id=\"tableRow\"><th><td id=\"userName\">User Name: " + localStorage.getItem('displayName') + "</td></th></tr>";
            table += "<tr id=\"tableRow2\"><th><td id=\"userEmail\">Email: " + localStorage.getItem('userLogin') + "</td></th></tr>";
            table += "</table>";
            infoDiv.innerHTML += "</br>";
            infoDiv.innerHTML += table;
        } else {
            infoDiv.innerHTML = "";
            var table = "<table id=\"table\" style=\" float: left; max-width:50%; max-height:100%; height:auto;\">";
            table += "<tr id=\"tableRow\"><th><td id=\"userName\">User Name: " + localStorage.getItem('displayName') + "</td></th></tr>";
            table += "<tr id=\"tableRow2\"><th><td id=\"userEmail\">Email: " + localStorage.getItem('userLogin') + "</td></th></tr>";
            table += "</table>";
            infoDiv.innerHTML += table;
            infoDiv.innerHTML += "<img id=\"img\" src=\"${pageContext.request.contextPath}/resources/images/noImage.jpg\" style=\"float: right; min-width: 15%; max-width: 30%; max-height:

        }
    });
</script>
```

Figure 24: Javascript code showing how the account page auto-adjusts layout based upon screen size.

8. Challenge Request Page

The challenge request page allows users to accept a friend's challenge request and also provides feedback to users on who the winner was of their completed challenges. This page can be accessed by pressing the menu icon in the upper left hand corner of any other page in QuizASSIST and then selecting the "Challenge Requests" header.

9. Friends Page

The Friends page allows users to add, delete, and view their friends. In addition, the page allows users to quickly look up friends to challenge, providing another way for users to challenge their friends.

MARKETING THE APP

During B Term the content group worked heavily on marketing the QuizASSIST application around campus as well as getting student feedback. It was crucial to gain the attention and support from the student body for the first round of pilot testing for this app. The flyers shown in Figure 8 below were used as paper advertisement that the content group distributed in highly concentrated areas on campus. These areas included the campus center, freshman dorms, the Morgan dining hall, and various lecture hall buildings.

Figure 25: QuizASSIST Flyer

When students caught wind of our new app, the next step was to reserve table sitting sessions to allow these students to play and give their feedback on what is done well and what can be improved upon. After the students completed at least one session on an iPhone or iPad, they were prompted to answer the following survey questions:

- Have you taken chemistry here at WPI before?
- Do you find that your tools or options for learning material are limited?
- How willing are you to try new tools for learning?
- Was this app easy to use?
- Did you like the color templates? Any suggestions?
- Was the content parallel to the content taught in the entry level courses at WPI?
- What were your likes or dislikes on the app?
- Do you have any suggestions for the app?

Through this first round of testing the content group expected to get information directly from the targeted market on how to improve the QuizASSIST app to meet the needs of students. In analyzing the results from this survey it was found that 90% percent of respondents believed that this app was easy to use. Another 60% of respondents found that the content in this application was parallel to the content taught in entry level chemistry courses at WPI. In this survey, it was brought to the attention that there was more than expected dissatisfaction with the user interface. Almost 25% of respondents commented that the color scheme was inconsistent or distracting. An important lesson learned here was that having a concrete color scheme is crucial because too many colors or clashing colors can be distracting for the user. This important note was taken

directly to the coding group to make immediate changes for the next version of the application. It is also important to note that the majority of students did not feel as though they needed alternative modes of studying and were indifferent to trying new tools of learning. This meant that getting students to use this app while taking the corresponding chemistry class may prove to be difficult or require more persuasion than thought originally.

After analyzing and understanding the points above, the content group passed along this information to the coding group to update the application according to the user needs and suggestions. Because the following term (C Term) would strictly be a testing phase, it was important that the content group secure the permission from at least one of the three professors teaching CH 1010 to promote the app. After reaching out to all three professors, one gave his permission to the content group to present the QuizASSIST app to his students.

Implementation in classroom

With the permission of Professor Burdette, the students enrolled in his CH 1010 course were used in this second round of pilot testing for this application. During the first week of the new semester, the content group set out to promote QuizASSIST in the classroom. The first step taken was introducing ASSISTments and its reputation to the students in this course. Through a short presentation, the content group provided the students with the main data point that the ASSISTments method has the ability to yield results such as 75% more learning than in a typical year[2]. It was important to make it clear to the students that they had all the control. With this application, these students

would have access to both relevant and recent study material to prepare for any assignment or exam they may have. At the end of the presentation students were able to put their name and email on a sign-up sheet to be emailed a downloadable version of the QuizASSIST application in the near future (see the Appendix D for the full list of emails).

When the 7 week pilot testing period was coming to an end, it was important to collect data on how this application worked for the students and what suggestions they may have. In order to analyze this information, the content group sent out a survey. The survey included the following questions:

- Did you find the content in the app parallel to the content taught in your chemistry course?
- How often did you use this app to study?
- Was this app a helpful way to study?
- What words would associate with this app?
- Was this app easy to use?
- What were your likes or dislikes with this app?
- Do you feel that this app was missing any components?
- Did this app encourage you to use challenge mode to study with your peers?
- Do you believe that this app helped you improve your knowledge for the course of chemistry 1010?
- Would you use this app if it had content for other courses?

- Would you recommend this app to a friend?
- How satisfied are you with the look and feel of this software?
- Do you have any thoughts on how to improve this software?

With the second round of testing, comments on user interface became more positive. This time, respondents used words such as simple and clean to describe the overall color scheme and user interface. The most relevant result found was information on when and how students used this applications. 50% of respondents recorded that this app was not used regularly (consistently once a week or more). Rather, this app was mostly used when there was an exam coming up and the students wanted to practice a variety of questions. The students reported that this way QuizASSIST allowed them to focus and prepare faster for the exam. This statistic aligns with one of the goals behind QuizASSIST, which is to provide an effective way of studying with a quick and easy use. Now, through the second round of testing, both the content and coding groups had a clear understanding of how the targeted market uses this application and what they expect moving forward.

EVALUATING GRAPHICAL USER INTERFACE

In the final weeks of developing this application the coding group made greater strides for QuizASSIST. The coding group added a web view for QuizASSIST, allowing students to study from a computer or laptop, as well as directly from the internet on a cell phone. During this process, the content group was responsible for making sure that the overall graphical user interface (GUI) and content was compatible with the new

changes. It was also crucial to make sure that the “challenge mode” and “friend requesting” features ran fluently and smoothly. During this time the content group had to focus on getting the feedback, from the students, in order to evaluate and improve the final GUI..

To ensure this, a third round of pilot testing was held in the form of small focus groups. In these focus groups the content group observed as students used both practice and challenge mode. In the rounds of practice mode, it was discovered that many students had trouble deciphering what it looked like when questions were labeled correct or incorrect. The top feedback given was darkening the green for correct and the pink for incorrect. This was due to bad contrasting between the colors as shown in Figure 27 below.

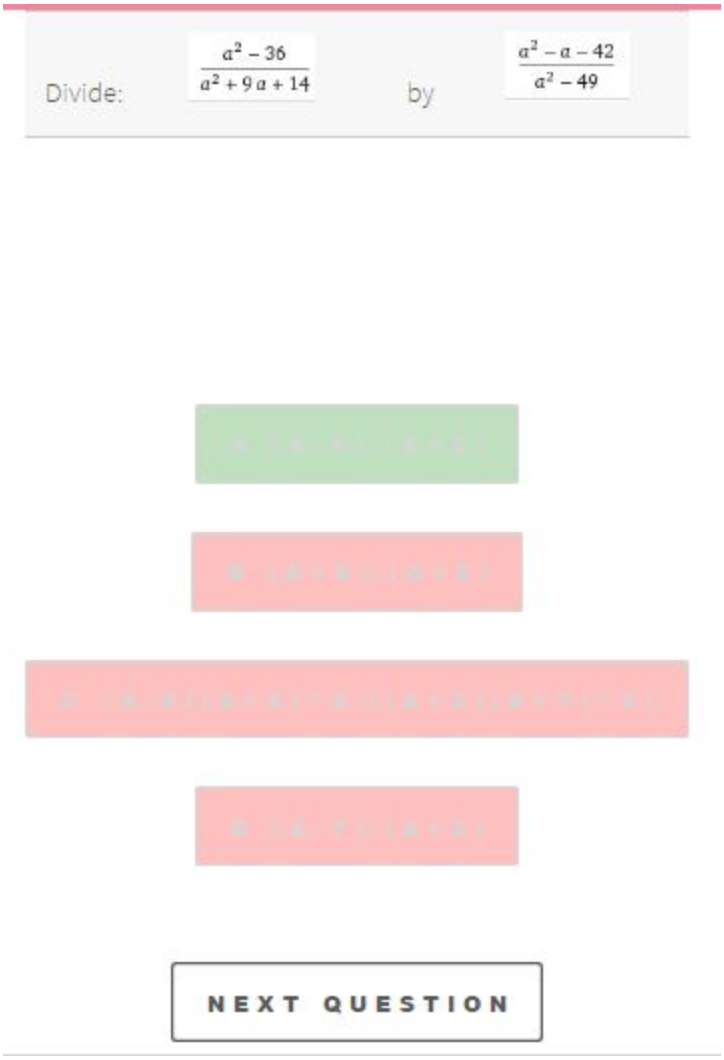


Figure 26: Multiple Choice Feedback

The students also noted that it would be helpful to add some sort of a message to highlight the correct or incorrect feedback. The student's feedback was passed to the coding group to make the appropriate changes. The final results are shown in Figure 28 below.

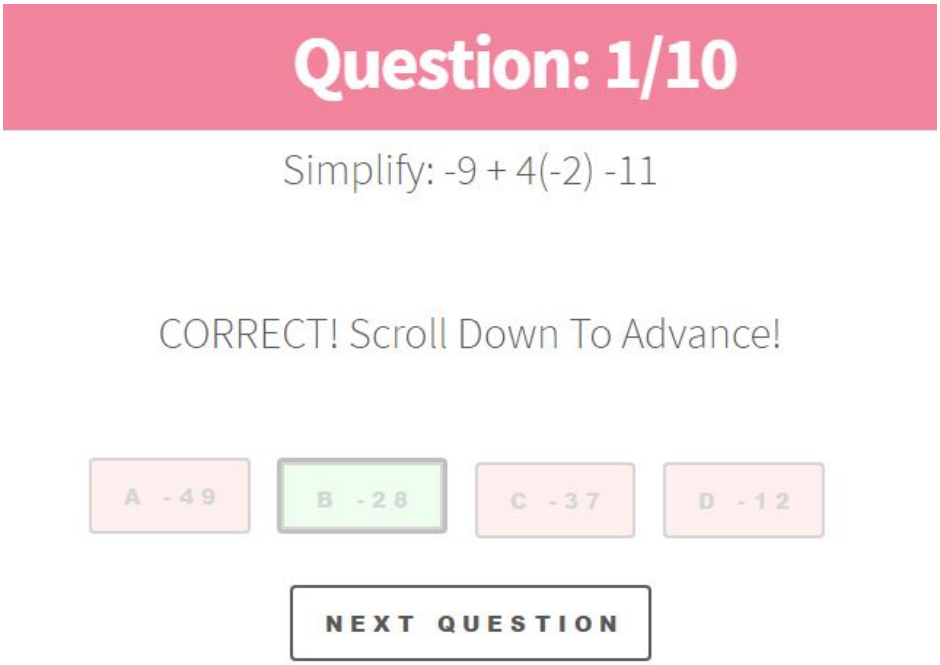


Figure 27: Final GUI for Multiple Choice Questions

Next, students were asked to go through the process of adding another friend and then participating in a challenge. After understanding how the basic practice mode works, this process for students was deemed to be fairly easy. Students figured to go to the "my friends" page and added a friend. Then they were either prompted to challenge this friend from the app, or go to the "challenge requests" page to accept a challenge or see the results of a previous challenge.

CONCLUSION

In conclusion, the QuizASSIST IQP team has developed a mobile application that can be used universally across any platform with the capability to connect to the internet. This app allows problems created on the ASSISTments website to be used by students to either practice on their own or to challenge other friends in a game-like setting.

For the future, the group has a few recommendations for how QuizASSIST should be handled. The first recommendation is to implement the “study group” mode. This mode involves a group of students working on the same problem sets together. In order to form the study group all the students must join and participate during the same time period. The idea is to have the questions delivered synchronously to all the participants. Students that answer correctly are able to give hints to other students that have answered incorrectly and have requested hints. Only after everyone has answered correctly the study group can move onto the next question. This “study group” mode will ultimately serve as a virtual study session for these students.

The next recommendation is to extend the targeted market of students for QuizASSIST to high school students.Towards the end of the project, the content group created a “Mathematics” folder with content pulled from ASSISTments. This addition to the content was made with a purpose to deliver a more universal type of problem content for pilot testing.

The group received positive feedback with this new subject and found that it could be a great outlet for students looking to refresh their memory on a variety of math topics before entering college. QuizASSIST can serve as a quick and easy way to practice and stimulate the brain before heading into a new setting for these students.

REFERENCES

[1] "Worcester Polytechnic Institute - Profile, Rankings And Data". *US News Best Colleges*. N.p., 2017. Web. 30 Apr. 2017.

[2] J. Roschelle, M. Feng, R. Murphy and C. Mason, "Online Mathematics Homework Increases Student Achievement", *AERA Open*, vol. 2, no. 4, p.1-12, 2016.

APPENDIX A: User Stories

During both A and B term, the team used a simple Google Doc to organize user stories and objectives. While this was not the most effective strategy, it initially worked due to the low number of developers on the project. In D term the number of developers increased substantially, so Trello, a software used to organize user stories, was used to make it easier to assign tasks to specific people. Below are images of all user storyboards that this IQP used.

A term - Version 1

User should be able to

- Start a new game for practice only
- Add and delete friends

Tasks for coding group

- Make the app support these problem types: multiple choice, check all that apply, exact match text answer (ignore case or not), numeric and algebraic type question (all kinds)
  (*according to assistments question types)
- Make sure html codes and images can be displayed properly
- New icons, logos and launcher image work on different screen sizes
- Remove redundant codes and databases and saved for future
- Publish the app for testing by random users
- User should be able to leave a game at any time - should not need to finish a game to leave

Tasks for content group

- Generate some organized problem sets into assistments database
- Generate at least 20 questions for some problem sets
- For each problem type, generate some questions for testing on the app
  (MC & CA : >=4 answers for each problem, only text in answers)
- Images and specified text styles are allowed
- Hints and feedback can be put in and used in the future

Figure 28: User stories and objectives for A-term

B term - Version 2

User should be able to

- Start a new game with a friend or stranger

Tasks for coding group

- Add challenge mode for game: user selects to play with a friend (choose a friend from friend list) or stranger; server side sends the same problems to both players
- Show notification when receiving a challenge; challenge has to be completed in some time; show whether opponent completed the challenge; when both players complete the challenge, update the status and show scores
- Show the score history (exact numbers or percentages, different game modes) to user
- Publish the app to app store
- Save progress on a particular practice quiz if the user left the game early

Tasks for content group

- Generate enough questions to be ready for use in C term

Figure 29: User stories and objectives for B-term

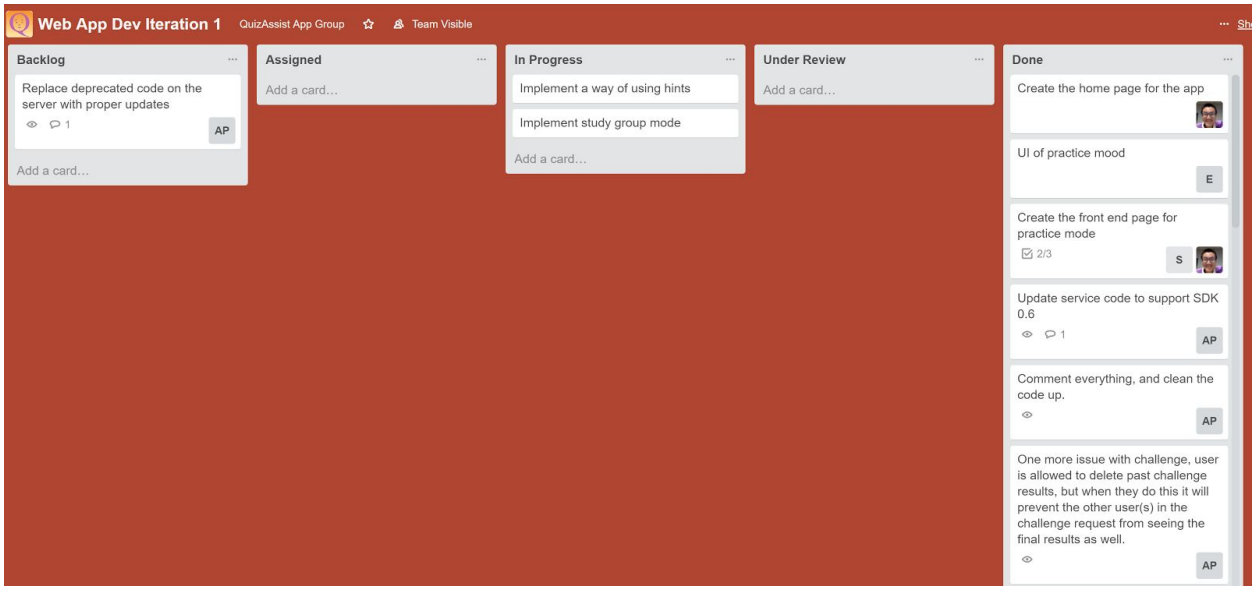


Figure 30: Trello storyboard to organize development objectives for D-term

APPENDIX B: Source Code From Current Project

Most of the initial Objective-C source code was decommissioned to account for the migration of the application to javascript. The code that was not decommissioned was converted to Apple's preferred programming language, Swift, once the migration had been completed. The application currently consists of 11 javascript files, 10 JSP pages, 3 CSS styling sheets, 28 Java files, and 2 swift files. All files can be accessed using SVN, contact the ASSISTments office for more details on how to do this. Several sample files are provided below.

1. Sample Javascript Source Files

The below code is taken from the gameTableViewController.js and the challengeRequests.js files only. This is done to show how both the practice and challenge modes work. For full documentation refer to the ASSISTments Fusion page.

```
/**
 * @author Andrew Petit
 * @description the below code is used to determine which question type the user is viewing, and
 * modifies the view based upon the question type
 */
//check if the question type is a multiple choice/check all or a fill in question
if ((questionJSON.problemType == "Multiple Choice") || (questionJSON.problemType == "Check All That Apply")){
    //there is a potential for more than four answers, we only want to show the user four options so we call this method to do that
    //the method returns a list of all the correct answers
    var correctAnswers = generateFourAnswers();


    //dynamically create the buttons Set the text of each button to be one of the answers.
    selectAnswer.innerHTML += "<button id=\"a0\" style=\"float: left;\" onclick=\"nextQuestionMC(correctAnswers, 0, 'a0')\">" + questionJSON.answers[0].answerText + "</button> ";
    selectAnswer.innerHTML += "<button id=\"a1\" style=\" \" onclick=\"nextQuestionMC(correctAnswers, 1, 'a1')\">" + questionJSON.answers[1].answerText + "</button> ";
    selectAnswer.innerHTML += "<button id=\"a2\" style=\" \" onclick=\"nextQuestionMC(correctAnswers, 2, 'a2')\">" + questionJSON.answers[2].answerText + "</button> ";
    selectAnswer.innerHTML += "<button id=\"a3\" style=\" \" onclick=\"nextQuestionMC(correctAnswers, 3, 'a3')\">" + questionJSON.answers[3].answerText + "</button> ";
    selectAnswer.innerHTML += "</br>";
    selectAnswer.innerHTML += "</br>";
} else { //this is the case when there is a fill-in question
    //dynamically add a text box so that a user can answer the question
    selectAnswer.innerHTML += "<form style=\"float: left;\" id=\"answerText\" onsubmit=\"return false\">" +
            "<input placeholder=\"Answer\" id=\"input\" type=\"text\" name=\"Answer\">" +
            "</form>  " +
            "<button id=\"checkButton\" onclick=\"preformCheck();\" style=\"float:right;\">Submit</button>";

    //preform an ajax request to send the response of the user to the server to check if they got the right answer or not
    $(document).ready(function() {
        $('#input').keyup(function(event) {
            if (event.keyCode == 13){//user pressed the enter key - so we should submit the answer
                var button = document.getElementById('checkButton');
                preformCheck(); //send an ajax request to determine if answer is correct
            }
        })
    })
}
```

Figure 31: The above code is a small example of the code used to display questions to a user. The above code differentiates multiple choice/check-all type questions from fill-in questions to allow for further processing.

```
 * @description makes an ajax request to the server to determine if check all response is correct
 * --going forward we can probably decommision some above and instead call this method
 */
function preformCheck(){
    var input = document.getElementById('input');
    var button = document.getElementById('checkButton');
    var inputValue = input.value;
    var response = $.ajax({
        url: checkFillIn,
        type: "POST",
        headers: {"Content-Type" : "application/json", "assistments-auth" : "partner=\"mobileAppService\"", "gzip" : "Accept-Encoding"},
        data: JSON.stringify({
            "problemId" : questionJSON.id,
            "response" : input.value
        }),
        success: function(data) { //if the ajax request passed
            console.log(data);
            var correctAsNum = 0;
            if (data.responseIsCorrect == true){//if the user got the right answer, we will add a point
                var num = parseInt(localStorage.getItem('numCorrect'))
                num += 1;
                localStorage.setItem('numCorrect', num);
                var answerArray = localStorage.getItem('correctAns');
                answerArray = JSON.parse(answerArray);
                answerArray.push(question);
                localStorage.setItem('correctAns', JSON.stringify(answerArray));
                input.disabled = true;
                input.style.backgroundColor = 'rgba(0,255,0,0.25)';
                correctAsNum = 1;
                correctDiv.innerHTML += "<p>CORRECT! Scroll Down To Advance!</p>";
            } else { //if not we will simply mark that the user got the question wrong
                var incorrectArray = localStorage.getItem('incorrectAns');
                incorrectArray = JSON.parse(incorrectArray);
                incorrectArray.push(question);
                localStorage.setItem('incorrectAns', JSON.stringify(incorrectArray));
                input.disabled = true;
                input.style.backgroundColor = 'rgba(255,0,0,0.25)';
                correctDiv.innerHTML += "<p>INCORRECT! Scroll Down To Advance!</p>";
            }
            logUserResponse(data.correctAnswer, input.value, correctAsNum);
            selectAnswer.innerHTML += "<button id=\"proceed\" style=\"display:block; margin:auto;\" onclick=\"nextQuestion()\">Next Question</button>";
            checkButton.disabled = true;
        },
        error: function() { //if the ajax request failed, we get here.
            console.log(this.url);
            console.log("fail");
        }
    })
```

Figure 32: The above figure shows how the program uses an ajax request to call a rest service provided by the Server. This ajax request allows the program to determine whether a fill-in response was correct or incorrect.

```
/**
 * @author Andrew Petit
 * @param correctAnswers
 * @param buttonPressed
 * @param elementId
 * @description determines if the question is a multiple choice or check all, then determines if the user got the question correct or incorrect and sends feedback to the user
 */
function nextQuestionMC(correctAnswers, buttonPressed, elementId){ //for multiple choice, we manually check answers on the client
    var b1 = document.getElementById('a0');
    var b2 = document.getElementById('a1');
    var b3 = document.getElementById('a2');
    var b4 = document.getElementById('a3');
    if(questionJSON.problemType == "Multiple Choice"){//multiple choice question
        if (buttonPressed == correctAnswers[0]){//user got right answer - update that the user got this answer right
            var num = parseInt(localStorage.getItem('numCorrect'));
            num += 1;
            localStorage.setItem('numCorrect', num);
            var answerArray = localStorage.getItem('correctAns');
            answerArray = JSON.parse(answerArray);
            answerArray.push(question);
            localStorage.setItem('correctAns', JSON.stringify(answerArray));
            //log user answer
            var correctIndex = correctAnswers[0];
            var correctResponse = questionJSON.answers[correctIndex].answerText;
            logUserResponse(correctResponse, correctResponse, 1);
            correctDiv.innerHTML += "<p>CORRECT! Scroll Down To Advance!</p>";
        } else {//user got wrong answer - update that the user got this answer wrong
            var incorrectArray = localStorage.getItem('incorrectAns');
            incorrectArray = JSON.parse(incorrectArray);
            incorrectArray.push(question);
            localStorage.setItem('incorrectAns', JSON.stringify(incorrectArray));
            //log user answer
            var userAnswer = questionJSON.answers[buttonPressed].answerText;
            var correctIndex = correctAnswers[0];
            var correctResponse = questionJSON.answers[correctIndex].answerText;
            logUserResponse(userAnswer, correctResponse, 0);
            correctDiv.innerHTML += "<p>INCORRECT! Scroll Down To Advance!</p>";
        }
        //we need to do it this way because of check all
        //essentially just update all the colors of every buttons
        changeButtonColors(buttonPressed, correctAnswers, "mult");
        selectAnswer.innerHTML += "<button id=\"proceed\" style=\"display:block; margin:auto;\" onclick=\"nextQuestion()\">Next Question</button>";
```

Figure 33: The above code displays the function that provides feedback to the user whether a multiple choice question they answered was correct or not. It also sends their answer back to the server to record the response so that feedback can be provided to the teacher. This same function also updates the display and logs all responses for check-all that apply questions, but has been omitted for simplicity - refer to the ASSISTments fusion page for full documentation.

```
/**
 * @author Andrew Petit
 * @param actualAnswer
 * @param userResponse
 * @param correct
 * @description sends a request to the server to log the response of the user, whether their answer was right or wrong, and other such data
 */
function logUserResponse(actualAnswer, userResponse, correct){
    console.log(localStorage.getItem('displayName'));
    console.log(localStorage.getItem('userId'));
    console.log(questionJSON.id);
    console.log(questionJSON.problemSetId);
    console.log(actualAnswer);
    console.log(userResponse);
    console.log(correct);
    var sendResponseToServer = $.ajax({
        url: logUserResponseURL,
        type: "POST",
        headers: {"Content-Type" : "application/json", "assistments-auth" : "partner=\"mobileAppService\"", "gzip" : "Accept-Encoding"},
        data: JSON.stringify({
            "userDisplayName" : localStorage.getItem('displayName'),
            "userId" : parseInt(localStorage.getItem('userId')),
            "problemId" : parseInt(questionJSON.id),
            "problemSetId" : parseInt(questionJSON.problemSetId),
            "actualAnswer" : actualAnswer,
            "response" : userResponse,
            "correct" : correct
        }),
        success: function(data) {
            console.log("success");
            console.log(data);
        },
        error: function(){
            console.log("fail");
        }
    })
}
```

Writable | Smart Insert | 292

Figure 34: The above code shows how logging is done for each problem answered by a user. An ajax request is made to the server to record a userDisplayName (email), a user ID, the problem ID, the problem set ID, the actual answer, and the user response in the database.

```
var request = $.ajax({
    url: getChallengeSetURL,
    type : "POST",
    headers : {
        "Content-Type" : "application/json",
        "assistments-auth" : "partner=\"mobileAppService\"",
        "gzip" : "Accept-Encoding"
    },
    data : JSON.stringify({
        "userLogin" : localStorage.getItem('userLogin')
    }),
    success: function(data){
        console.log(data);
        var checkRequests = 0;
        var checkResults = 0;
        if (data.message.length != 0){ //user has friends
            for (var i = 0; i < data.message.length; i++){ //add a checklist of all the user friends to let them edit the contents of their friends and challenge multiple people
                if (data.message[i].status == 0){//pending request
                    //should include an if statement to check status of problemset - if set to 2, challenge was done - if set to 3, challenge was deleted
                    challengeList.innerHTML += "<li class=\"list-group-item\" onclick=\"acceptChallenge("+ data.message[i].id  +")\">"+
                    data.message[i].friendDisplayName + "'s Challenge" +
                    "</input></li>";
                    checkRequests = 1;
                }

                if (data.message[i].status == 1){
                    if(data.message[i].winnerLogin == userLogin) {
                        resultsList.innerHTML += "<li class=\"list-group-item\"><input type=\"checkbox\" name=" +
                        data.message[i].id +">You beat " +
                        data.message[i].friendDisplayName +
                        "!</li>";
                    } else {
                        resultsList.innerHTML += "<li class=\"list-group-item\">" +
                                "<input type=\"checkbox\" name=" +
                                data.message[i].id +
                                ">You lost to " +
                                data.message[i].friendDisplayName +
                                "! Better luck next time!</li>";
                    }
                    checkResults = 1;
                }
            }
        }

        if (checkRequests == 0){
            challengeList.innerHTML += "<li class=\"list-group-item\">No Current Challenge Requests</li>"; //might not want this - kind of mean
        }
```

Figure 35: The above code shows how the application pulls challenge requests for certain users. In essence, a server call is made which looks at the database for any challenge requests for the user with a status of 1. This status indicates that a challenge is pending, and the user can then click on the challenge request to take it.

2. Sample JSP Pages

The following JSP/HTML code is taken from index.jsp, challengeRequests.jsp, and games.jsp. All other JSP/HTML code can viewed on the ASSISTments Fusion page.

```
<!-- Main -->
    <div id="main">
        <div class="inner">
            <header>
                <h1>Welcome to QuizASSIST, a study game for students.</h1>
            </header>
            <section class="tiles">
                <article class="style1">
                    <span class="image">
                        <img src="${pageContext.request.contextPath}/resources/images/pic01.jpg" alt="" />
                    </span>
                    <a href="questions.jsp">
                        <h2>Practice</h2>
                        <div class="content">
                            <p>This is practice mode which will help you prepare for an exam.</p>
                            <P>Use it to ace the class!</P>
                        </div>
                    </a>
                </article>
                <article class="style2">
                    <span class="image">
                        <img src="${pageContext.request.contextPath}/resources/images/pic02.jpg" alt="" />
                    </span>
                    <a href="challenge.jsp">
                        <h2>Challenge</h2>
                        <div class="content">
                            <p>This is challenge mode which allows you to compete with your friends.</p>
                            <p>See who is the best!</p>
                        </div>
                    </a>
                </article>
                <article class="style3">
                    <span class="image">
                        <img src="${pageContext.request.contextPath}/resources/images/pic03.jpg" alt="" />
                    </span>
                    <a href="studygroup.jsp">
                        <h2>Study Group</h2>
                        <div class="content">
                            <p>This mode allows you to create a virtual study group with your friends and study together for an exam.</p>
                            <p>Studying will never be lonely again!</p>
                        </div>
                    </a>
                </article>
            </section>
        </div>
    </div>
```

Figure 36: The above code shows how the group used the HTML framework to show build the three main boxes on the opening index.jsp page.

```
</br>
    <li style="display:table; margin:auto;">
        <div class="panel-group" id="accordion">
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h4 class="panel-title">
                        <a data-toggle="collapse" data-parent="#accordion" href="#collapse1">Choose a Challenge to Accept!</a>
                    </h4>
                </div>
                <div id="collapse1" class="panel-collapse collapse">
                    <ul class="list-group" id="challengeList"></ul>
                </div>
            </div>
        </div>
    </li>

    <li style="display:table; margin:auto;">
        <div class="panel-group" id="accordion">
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h4 class="panel-title">
                        <a data-toggle="collapse" data-parent="#accordion" href="#collapse2">Challenge Results!</a>
                    </h4>
                </div>
                <div id="collapse2" class="panel-collapse collapse">
                    <ul class="list-group" id="challengeResults"></ul>
                </div>
            </div>
        </div>
    </li>
```

Figure 37: The above code is taken from challengeRequests.jsp. This code shows how the team built the drop down boxes to display the challenge requests.

```
<body>
    <header>
        <div id="headerColor" style="color: white; padding: 20px; height: 100px;">
        <script src="${pageContext.request.contextPath}/resources/js/changeModeColor.js"></script>
        <button style="float: left;" onclick="location.href = 'questions.jsp';">X</button> <!--Possibly very unnecessary, but would be nice for the mobile views -->
        <center><h1 id="modeHeader">Question: </h1></center>
            <header id="header">
                <nav>
                    <ul>
                        <li><a href="#menu">Menu</a></li>
                    </ul>
                </nav>
             </header>
        </div>

        <!-- Menu -->
        <nav id="menu">
            <h2>Menu</h2>
            <ul>
                <li><a href="index.jsp">Home</a></li>
                <li><a href="account.jsp">My Account</a></li>
                <li><a href="friends.jsp">My Friends</a></li>
                <li><a href="challengeRequests.jsp">Challenge Requests</a>
                <li><a onclick="javascript:logOut();">Log Out</a></li>
            </ul>
        </nav>
    </header>

<link rel="stylesheet" href="${pageContext.request.contextPath}/resources/stylesheet/questionFormat.css"/>
<div id="answer" style="display: table; margin: auto;"></div>
<div id="correct" style="display: table; margin: auto;"></div>
<div id="selectAnswer" style="display: table; margin: auto;"></div>
<script src="${pageContext.request.contextPath}/resources/js/gameTableViewController.js"></script>
```

Figure 38: The above code is taken from game.jsp. While this is where the game actually occurs, there is not that much HTML code because a lot of the design changes are made directly in the javascript source file gameTableViewController.js.

3. Sample CSS Stylesheets

The following CSS code is taken from the stylesheet file, practice-question.css. All other stylesheets can be viewed on the ASSISTments Fusion page.

```
li {
    list-style-type: none;
}

div.container4 {
    height: 5em;
    position: relative
}
div.container4 p {
    margin: 0;
    position: absolute;
    font-family:   inherit;
    font-size:      80px;
    font-weight:    bold;
    top: 50%;
    left: 50%;
    margin-right: -50%;
    transform: translate(-50%, -50%)
}

 div.boxed {
    padding: 20px 20px 20px 20px;
}

/* Dropdown Button */
.dropbtn {
    background-color: #4CAF50;
    color: white;
    padding: 16px;
    font-size: 16px;
    border: none;
    cursor: pointer;
}

/* Dropdown button on hover & focus */
.dropbtn:hover, .dropbtn:focus {
    background-color: #3e8e41;
}

.dropdown {
    position: relative;
    display: inline-block;
}
```

Figure 39: The above code is taken from the practice-question.css page. This css stylesheet is used to style the questions.jsp page.

4. Sample Java Source Files

The following java code is used to set up the server and also create RESTful based web services that the application uses to update tables in the database. The following samples are taken from the logAnswerManagerImpl.java and logAnswerController.java source files, two files used to create a RESTful service that

logs user responses in a table on the database. Like with the rest of the provided samples, a full documentation including all java files can be found on the ASSISTments Fusion page.

```java
@Component
public class LogAnswerManagerImpl implements LogAnswerManager {

    @Autowired private UserResponseDao answerDao;

    @Override
    public Map<String, Object> logUserAnswer(String userDisplayName, int userId, int problemId, int problemSetId,
            String actualAnswer, String response, int correct) {
        Map<String, Object> feedback = new HashMap<String, Object>();

        try {
            List<QueryTerm> terms = new LinkedList<QueryTerm>();
            QueryTerm term1 = UserResponseDao.Field.USER_DISPLAY_NAME.getQueryTerm(userDisplayName);
            QueryTerm term2 = UserResponseDao.Field.USER_ID.getQueryTerm(userId);
            QueryTerm term3 = UserResponseDao.Field.PROBLEM_ID.getQueryTerm(problemId);
            QueryTerm term4 = UserResponseDao.Field.PROBLEM_SET_ID.getQueryTerm(problemSetId);
            QueryTerm term5 = UserResponseDao.Field.ACTUAL_ANSWER.getQueryTerm(actualAnswer);
            QueryTerm term6 = UserResponseDao.Field.RESPONSE.getQueryTerm(response);
            QueryTerm term7 = UserResponseDao.Field.CORRECT.getQueryTerm(correct);

            terms.add(term1);
            terms.add(term2);
            terms.add(term3);
            terms.add(term4);
            terms.add(term5);
            terms.add(term6);
            terms.add(term7);

            Answer answer = new Answer();
            answer.setUserDisplayName(userDisplayName);
            answer.setUserId(userId);
            answer.setProblemId(problemId);
            answer.setProblemSetId(problemSetId);
            answer.setActualAnswer(actualAnswer);
            answer.setResponse(response);
            answer.setCorrect(correct);

            answerDao.persist(answer);
            feedback.put("message", "Logged User Input");

        } catch(Exception e) {
            feedback.put("message", "Error occured. Please try again.");
            e.printStackTrace();
        }
        return feedback;
    }
```

Figure 40: The above code is taken from the logAnswerManagerImpl.java file. It highlights the method by which the server accesses the database and creates a new row in the UserResponse table with data provided in the parameters of the logUserAnswer method.

```java
package org.assistments.mobileservice.web;

import java.util.Map;

@Controller
public class LogAnswerController {

    @Autowired private LogAnswerManager loggingManager;

    @RequestMapping(method = RequestMethod.POST, value = "/logging/userResponse")
    public ResponseEntity<Map<String, Object>> logData(@RequestBody Map<String, Object> request){
        System.out.println("Processing request /logging/userResponse ...");

        Map<String, Object> temp = loggingManager.logUserAnswer((String)request.get("userDisplayName"), (int)request.get("userId"), (int)request.get("problemId"),
                (int)request.get("problemSetId"), (String)request.get("actualAnswer"), (String)request.get("response"), (int)request.get("correct"));
        return new ResponseEntity<Map<String, Object>>(temp, HttpStatus.OK);
    }
}
```

Figure 41: The above code is taken from the logAnswerController.java file. This code essentially sets up the RESTful service with the url ending in /logging/userResponse. When a JSON file is sent through an ajax request call, this controller parses the JSON file and calls the method logUserAnswer on the logAnswerManagerImpl.java file.

5. Swift Files

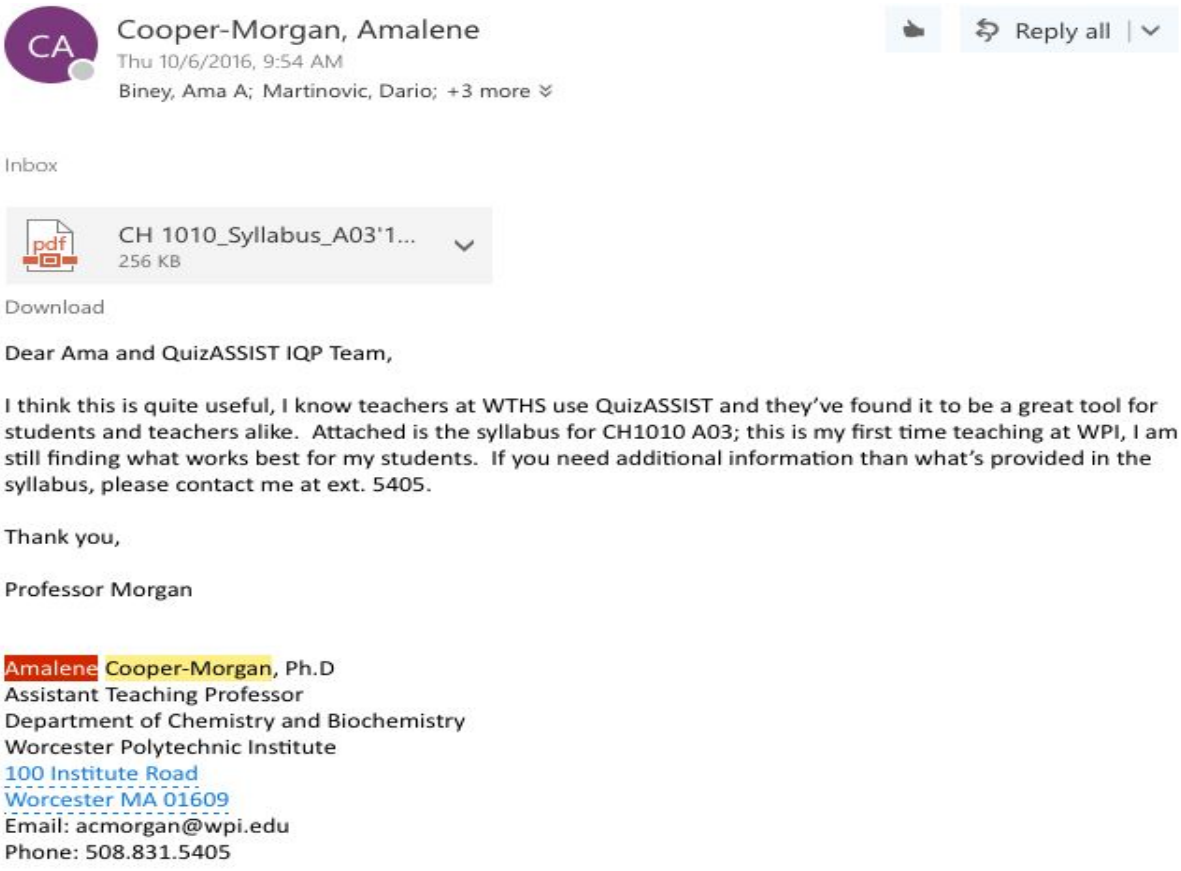
While all of the Objective-C code was decommissioned when migrating the application over to a hybrid mobile app based system, a single swift file has been added. This files essentially just work to set up the iOS application and open the application URL in a webview. The full documentation for this file is provided below.

```swift
//
//  WebViewController.swift
//  QuizAssist
//
//  Created by Andrew Petit on 4/5/17.
//  Copyright © 2017 WPI Assistments. All rights reserved.
//

import UIKit

class WebViewController: UIViewController {
    @IBOutlet var webview: UIWebView!

    override func viewDidLoad() {
        super.viewDidLoad()
        let urlAddress = "http://astest1.cs.wpi.edu:8080/quizassist/quizassist-web-template/index.jsp";
        let url = URL(string: urlAddress);
        let requestObj = URLRequest(url: url!);
        self.webview.loadRequest(requestObj);
        self.view.addSubview(webview);

        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

}
```

Figure 42: The above code is taken from the webViewController.swift file. It works to instantiate a webview that will open to the index.jsp page of the hybrid mobile application based system.

APPENDIX C: Permission Emails From Professors

In order to understand what content was being taught in chemistry courses and how this content was being taught, the group needed permission from current professors to get this insight. The group reached out to four professors, two kindly declined, while two others gave us access to their syllabi and/or added the group as observers to the course website on canvas. These are their responses.



CA

Cooper-Morgan, Amalene

Thu 10/6/2016, 9:54 AM

Biney, Ama A; Martinovic, Dario; +3 more

Reply all

Inbox

pdf CH 1010_Syllabus_A03'1...

256 KB

Download

Dear Ama and QuizASSIST IQP Team,

I think this is quite useful, I know teachers at WTHS use QuizASSIST and they've found it to be a great tool for students and teachers alike. Attached is the syllabus for CH1010 A03; this is my first time teaching at WPI, I am still finding what works best for my students. If you need additional information than what's provided in the syllabus, please contact me at ext. 5405.

Thank you,

Professor Morgan

Amalene Cooper-Morgan, Ph.D
Assistant Teaching Professor
Department of Chemistry and Biochemistry
Worcester Polytechnic Institute
100 Institute Road
Worcester MA 01609
Email: acmorgan@wpi.edu
Phone: 508.831.5405

Figure 43: Email expressing Professor Morgan's interest in helping the QuizASSIST team with creating chemistry content.

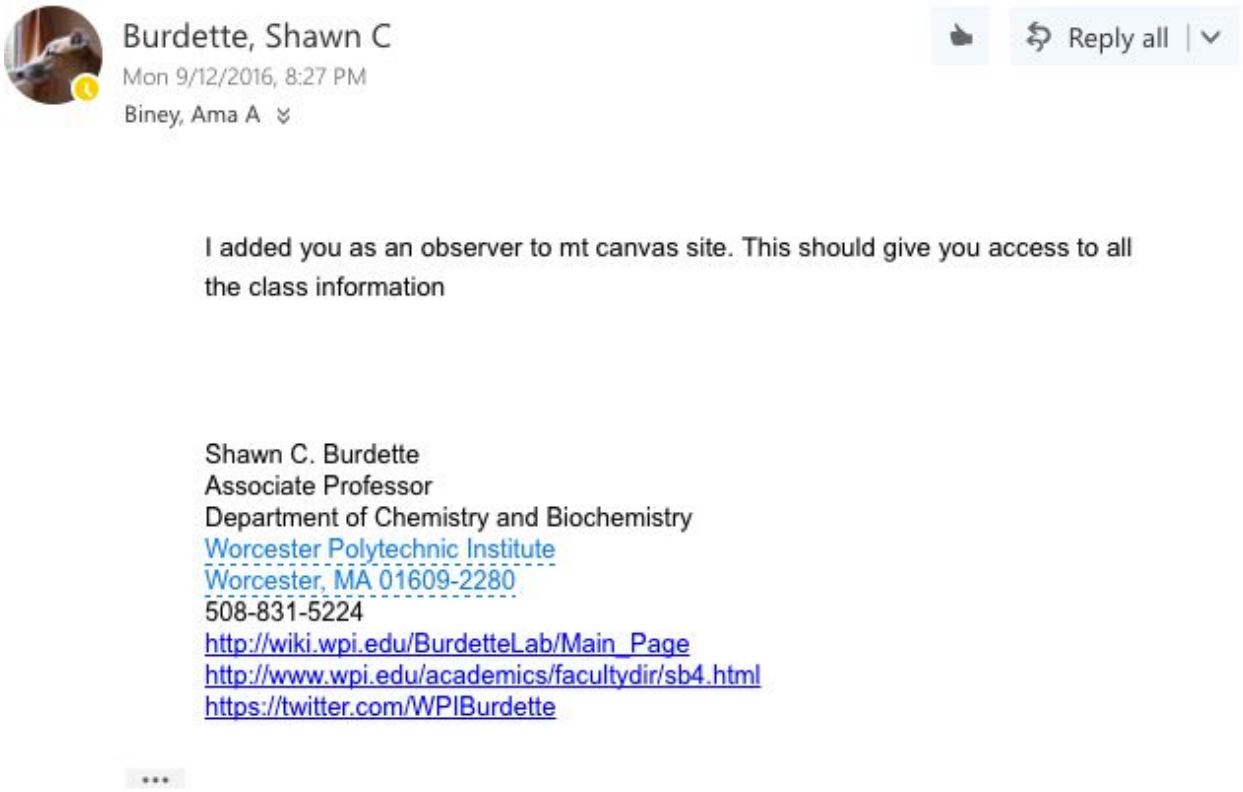


Figure 44: Email expressing Professor Burdette's interest in helping the QuizASSIST team with creating chemistry content.

APPENDIX D: QuizASSIST C-term Interest Emails

C-Term Interest List of emails
caladin@wpi.edu
cjrodgers@wpi.edu
jgirgenrath@wpi.edu
mrmcfatter@wpi.edu
cteng@wpi.edu
sburns@wpi.edu
msalamone@wpi.edu
aptoledobarrios@wpi.edu
cmholmberg@wpi.edu
abeshir@wpi.edu
cmfiddes@wpi.edu
slor@wpi.edu
apesce@wpi.edu
ammaffeo@wpi.edu
dtbarber@wpi.edu
csonderson@wpi.edu
mfkesten@wpi.edu

Figure 45: A List of a few students that signed up to download the QuizASSIST application off of Apple's distribution software, TestFlight.