Final Project Report

GAMING ED.

CAPSTONE 2022

Grace Hsieh, Brett Lewerke, and Chayson Spigarelli

15 December 2022

Project Sponsor: Terry E Baxter

Faculty Mentor: Dr. Michael Leverington

Table of Contents

1.0 Introduction

Team Gaming Ed. has built a learning management system specifically designed to increase learning motivation compared to other learning management systems. How the team is doing this is by gamifying our learning management system. By gamifying our learning management system the team can improve student motivation by making students feel like they are playing a game when completing school work.

Our client Professor Terry E Baxter currently gamifies one of his classes using the schools BbLearn learning management system. However this is tedious and our client spends a lot of extra time trying to gamify his course since it all has to be done manually. For example, in order to track students' coin counts our client has to track them all manually in an Excel spreadsheet. The goal of Team Gaming Ed. is to eliminate extra manual work done by our client in order to save him time and for the students to have a more truly gamified learning management system.

2.0 Process Overview

Grace was our team leader and helped the team stay on top of project deadlines and ensure that the team was submitting top-level deliverables. Brett was our team architect and had some previous work experience using .NET Framework which was extremely beneficial to the team as it helped the team get on the right foot. Chayson was the team's task report manager. Chayson made sure that the team was aware of projects coming up as well as making sure everyone was completing their weekly sprints. Our client was the one who came to the team with the idea to build the RTX Gamification Learning Management System. Our client had tons of ideas he wanted for the gamified learning management system which helped drive development projects.

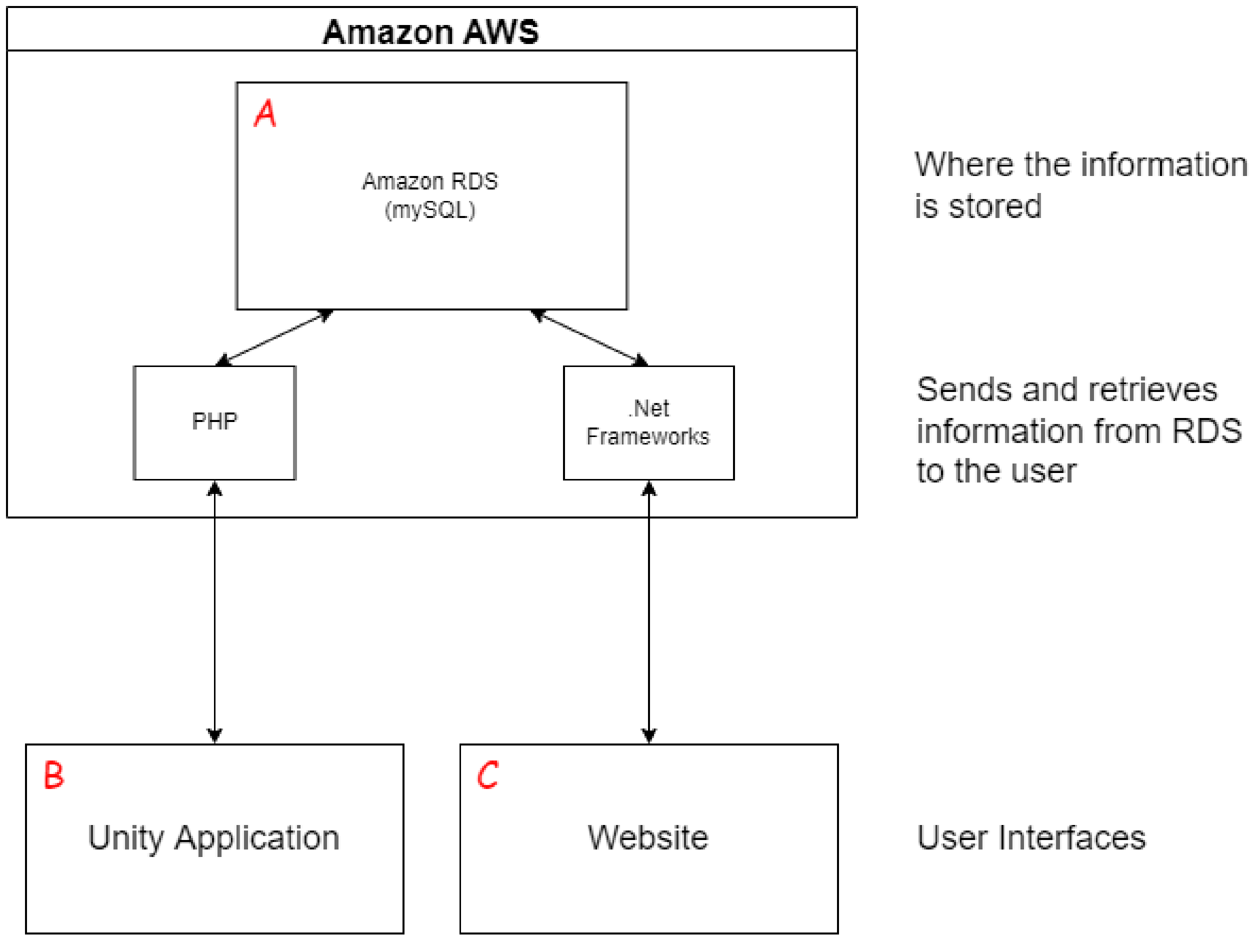
Many tools were used throughout the course of the project. For managing code the team used GitHub. For developing the website the team used Visual Studio 2022 and used its .NET Framework. For developing the desktop application the team used Unity Game Engine in a 3D environment.

3.0 Requirements

The team split the requirements into two different sections including functional requirements and non-functional requirements. The functional requirement define a system or its component while a non-functional requirement defines the performance attribute of a software system. The functional requirements of the system were administrator profile management, creating course content, managing existing course content, player profile management, and players interacting with course material. All of these are core functional requirements listed by the team before going into project development. On the other hand the non-functional requirements were ease of use, data integrity, speed, and scalability. These non-functional requirements all affected the overall performance of the system and needed to be thought about throughout the course of project development.

4.0 Architecture and Implementation

There are three major components of our project, the relational database (RDS) using MySQL query language [A], the Unity based desktop application [B], and the .NET Framework website [C]. The figure shows how they work together as a whole. This section will go into the key features and components, how information travels between them, and the comparison of what we expected to have and what ended up being reality.



4.1 Website Application

The team chose .NET Framework for many reasons. One of the most important reasons would be that the website needed to access the database all the time while writing normal HTML code. .NET Framework utilizes .razor pages which allow programmers to dynamically write HTML code. For example programmers can access the database alongside the HTML code in order to dynamically write HTML code. In this projects case the program accesses the database using C# and can be written alongside regular HTML code. This made accessing the database from the website application extremely easy. This was really important because we are accessing the database very frequently on the website.

4.1.1 Key Features and Components

Key features of the website include where in the code the team connects to the database. This is located in the *appsettings.json* file. Simply just replace the host name, port, schema, and password in order to connect to a different database. This is the only thing that you need to change in order to connect to a new database. Purely C# code is written on each page at the bottom inside the *code {}* block. All libraries are located at the top of each page beginning with the @ symbol followed by the library name.

4.1.2 Communication Mechanisms

Pages can communicate with each other by accessing variables from the database. For example if one page inserts data into the database another page can retrieve the data just submitted to the database. At the top of each file shows what the page is specifically called. So if you ever want to know how to get to another page you need to look at the very top of the file and use the identifier next to *@page.* To actually navigate to another page the programmer needs to use the Navigation Manager which is used by typing *NavManager.NavigateTo("/page").* In order to code the dynamic html documents, a *@code* will be needed to actually code the C# needed for the page. These C# functions may include accessing the database and configuring the html page layout. In order to actually use the C# code inside of the html pages, the user must assign the values taken from the database into a model. They then use the @ sign in the razor page to access the needed values to display to the user. These mechanisms given to us by .NET give us the ability to dynamically display html pages if the information exists and how much there is to display.

4.1.3 Original vs Outcome

The ability to code the website in .NET became a challenge for us as soon as we started coding it. One of our team members has previous experience with the framework so the team learned from what they coded for our first semester demo. The team definitely tried to bite off as much as we thought we could chew. For example the team wanted to include exemption medals and purchase new attempts. The team and client came together and fleshed out many ideas on how exactly the learning management system would be gamified compared to other learning management systems. In the end we had a ton of great ideas but just not enough time during the project development to complete all of these goals.

4.2 Desktop Application

The team needed a way to build the course in an offline environment and have it be in a 3D environment. With these restraints the team came up with a solution that tackles both of these problems. This section will outline how the desktop application works with the website application and every step in between.

4.2.1 Key Features and Components

The team decided to use the Unity game engine for this project because one of the project requirements was that building the course information needed to have an offline functionality. A website to create the course was not an option due to the project requirements. The Unity game engine uses a mix of it's own functionalities to build the UI and C# to handle the background processing. Unity uses files called scenes to load what the user is seeing. In this scene, you can attach C# code to an object and then execute different functions depending on what the user does. For example, a function in C# might be called "GoBack()" and it is never called during the rest of the file. A button that is linked to the script in the scene is the one that is actually in charge of executing this function and it executes the proper code inside of it. The team also used other technologies such as Blender to create custom 3D objects that are used in the game. Unity also uses a method called PlayerPrefs, which is a global variable and can be called anywhere throughout. game. We used these variables to pass information around between different scenes since Unity destroys variable information after a scene change. We decided to use this instead of classes because Unity does not like when you use the new() method to substantiate classes. Unity does not have the ability to access a database using its own libraries so PHP code was used as a sort of API to handle the information sent from Unity. The PHP code sits on a server (most likely the same one as the website) and it takes in information from a POST request that Unity sends. It is then in charge of formatting that information into SQL strings that is then sent to the database for the website to use as its information.

4.2.2 Communication Mechanisms

The Unity game engine is very good at being a video game and less so as a UI to input information. After each object's information is inputted by the user and the save button is clicked, it saves the file into a JSON file on your local computer. The name of this file is the location of the object you are saving, making it easy to respawn the objects with the correct information if the scene is changed. When a user would like to upload the course to the database, the C# code reads the JSON files in a folder that is the name of the class you are editing. It then formats each variable inside of the JSON document and puts it into a form and sends it to the PHP code. The PHP code takes in the information and formats it into a SQL string and updates the tables with the information. For simpler requests, like logging in, Unity only takes in the information that is recently typed and does not save it in a JSON file. The PHP code is in charge of handling the logic if the user has typed in the correct/incorrect login information.

Since Unity does not allow the passing of information from one scene to another, we decided to use a method called PlayerPrefs. In order to access the method you need to specify the type of variable you want saved with either SetInt() or SetString(). You then pass the name of the variable and the information you want it to hold. We used this saving mechanism to pass information like the course name, whether or not the user is logged in, or if the user wants to edit some object. The names of these variables must be unique or you will run into errors.

4.2.3 Original vs Outcome

When we first envisioned this project, we did not understand the complexity that goes into making an application in Unity. The learning curve for this engine is very steep and it took a month or so to just understand how Unity passes its information around. There are many built in functionalities that Unity recognizes that the average user looking at it would not understand. With that being said, I think that our Unity portion of this project was a huge undertaking and we did not completely tie off every loose end that came our way. There were many times throughout the semester that hundreds of lines of code were replaced with just a few lines once we knew that Unity had something already built for it. The documentation for Unity is very hard to understand and there are a million ways to do one thing. For example, when first designing the main course creation page (where you drag and drop the objects) it took hours of research to learn how to raycast from our mouse. We originally designed it so that when an object collides with a highlight it would get sucked to it and you did not have to click. However, since page refreshes so often the data from the collider would be destroyed and it did not work. We later understood that you can compare the tag of an object to another one in order for it to know when you hit a highlight. Also, learning how to differentiate between a local position and global position was also a huge challenge and it took a lot of refactoring of code before we got this to work. Before we started using PlayerPrefs, we tried to use a class that held the object's information. Little did we know that Unity destroys the information on a scene change and this would not work.
Some things that we did not include were solely due to the lack of time and manpower that we had during the project. Our original vision was that we could view an essay that the student inputted into the course, however Unity does not support displaying PDF's or any other file type. Also, our plan was to export the student grades into an excel spreadsheet for the teacher to easily

view but we did not have the time for this. Other ideas that did not make it into the project due to the same reasons was: changing a students grade manually, viewing past assignment history by date, adding avatars through unity, customizing object icons, having a working nonlinear course structure, deleting individual courses that teachers take, changing a teachers username and password, emailing students about updates and submitted assignments, and much more.

4.3 Database

The database is the central part of the project where all the data from both the website and the desktop application are stored and accessed. A good foundation was important in order to allow the website and desktop applications to function without needing to be held back by the data it was trying to use. This section will outline the process the team used to get the database to the state it is currently in.

4.3.1 Key Features and Outcomes

The database itself is written in MySQL, which is an RDS, relational database, capable of using and managing keys. This means that much more complicated variables can be formed from smaller parts. We did all of this within MySQL Workbench. This is another way to view and generate the code behind the database. The variables were formatted in such a way that collisions were less likely to occur when all team members were working at the same time. So although it may be more inconvenient to code, it allows us to keep track of what is happening and have some consistency with naming conventions.

Another reason to use a RDS is the keys. Primary keys are the identifiers of a row which can be used in a different table as a foreign key to tell the program that those two different rows in different tables are related to each other. Consequently, this allows for a larger network of variables to be associated with each other while also reducing the amount of redundantly saved information.

4.3.2 Communication Mechanisms

The desktop and the website applications both have their own way to interact with the RDS. However, the thing that stays the same is the core idea that a command is sent to the database formatted as an SQL string. Specifically, for the website string commands are passed to the database and the desktop application has built in PHP functions to do the same thing.

One thing that the database is able to do is create and store views. Views are multiple tables put together using the primary and foreign keys to concatenate the tables to reduce the number of potential calls to the database the program needs to do. The reason why the columns are named as they are is to facilitate the building of views using the wildcard symbol (*) to quickly put together multiple tables and not have to check for naming collisions.

4.3.3 Original vs Outcome

In the beginning, the team did not understand the idea of keys very well and so the project ended up with one large "mega table" to hold all of the information at the same time. After some large refactoring, the team ended up with the current model that is being used today. There are some notable redundancies, but too many iterations would have confused the team and created more work to get both the desktop application and the website to correctly use any new variables. A better planning phase would have prevented this which is good to know for the future.

5.0 Testing

Unit testing was pretty similar for both the website application and the desktop application. Different parts of the projects were tested individually for robustness. Integration testing was also pretty similar for both website and desktop application testing which tested how combined entities of the project flowed together. However with usability testing the desktop application needed teachers to test it because that would be who used the desktop application while for the website application the team needed students to test it because that is who would be using the website application.

5.1 Website Testing

Unit Testing: There were many different units of the project that the team needed to test in order to make sure that the system did not break. The assessments took the longest to test as there were many different factors that went into assessments including earning coins, due dates, multiple types of questions, and adding scores to the course gradebook.

Integration Testing: Integration testing was used to test multiple project components at once to see if they were all working cohesively. For example when completing an assessment a student should earn coins and be able to view their grade in the gradebook. So when testing after completing an assessment the developer would make sure the correct amount of coins were added to a students account and that their previous grade on an assessment was also displayed inside the students course gradebook.

Usability Testing: For testing the website the team decided that the best way to test would be to have students at NAU test the website since these are the users the website is designed for. The team simply had their friends/roommates test the website to make sure nothing on the website would break.

5.2 Unity Testing

Unit Testing: There were many different parts of the desktop application that needed to be tested individually. For example when creating an assessment the team needed to make sure that questions were saved properly so that the teacher could edit an assessment without having to restart from scratch. The most difficult unit to test was the offline functionality of the desktop application which allowed a teacher to create a course without an internet connection and then be able to upload that course once an internet connection was established.

Integration Testing: For the desktop application the team used integration testing to test multiple project components at once to make sure everything was working together properly without breaking. For example once a super-administrator deleted a teacher the teacher should not be able to be deleted again since it should be already gone. Also when the teacher who just got deleted tries to sign in again then they should not be granted access since the super-administrator deleted their account.

Usability Testing: For testing the desktop application the team decided that the best way to test would be to have teachers at NAU test the desktop application since these are the users the desktop application is designed for. Our client Professor Terry E Baxter was our main test case because he would be the main person using the desktop application.

6.0 Project Timeline

In order to get our project done in a timely manner, we did our coding sessions in weekly sprints. These weekly sprints assigned by Brett were decided during each meeting. However, the majority of the team needed to be okay with what the other team members were doing during their sprints. These sprints were done in a manner that would build off the previous week's sprint so that the team would be focusing on a portion of the project. The deadline for the project was the 2nd of December as this was the day that the team presented the demo during the capstone conference.

For Chayson:
His sprints were mainly designed around getting assignments and tests on the website, adding the course home page, and styling the website. These sprints had a set due date at the end of the week, however due to the coding issues these had to sometimes be extended further.
Throughout the semester he added the ability to:

Phase 1:
- Adding the main home page for course
- Adding a link to view discussion board for a course
- Adding a link to view the marketplace for a course
- Adding a link to view the levels for a course

Phase 2:
- View recently turned in assignments
- Complete a exam including the following types of questions
  - Multiple choice questions
  - Fill in the blank questions
  - Matching questions
- Show the grade that the student received on the exam
- Show how many points they earned after completing the exam
- Give the option to show questions that the student missed

Phase 3:
- Styling the header with NAU colors
- Styling the main home page and course page
- Styling the discussion board
- Styling the marketplace page
- Styling the test and assignment pages

For Grace:
Her sprints were involved around everything else about the website which included adding a discussion board, adding marketplace, adding the main home page, adding the student profile page, and creating the whole database. Sometimes these sprints would take longer than a week and needed to be extended.

Throughout the semester she added the ability to:

Phase 1:

- Create the tables needed for account information
- Create the tables needed to have infinite levels
- Create the tables needed to have infinite tests
- Create the tables for discussion board posts

Phase 2:

- Fixing the discussion board to view more information
- Replying to the discussion board
- Adding the ability to delete a discussion you posted
- Add the avatars into the website
- Adjust the table for coins
- Add the ability to purchase avatars

Phase 3:

- Add the link to view courses
- Create page to view the individual courses
- Add student profile page

For Brett:

His sprints were for the Unity project only as Brett only worked on the Unity portions of the project. Since he did not have another teammate working on the project, his sprints were done out of order. His portion of the project was large and sometimes the sprints would slur together during the weeks and he completed as much as he could. Since there were really no deadlines besides the 2nd of December, he did not have any phases during his sprints. During each team meeting he would show off to the team what he did during the week and they would either approve or disapprove. The team would offer insight on what they would want to see done next.

7.0 Future Work

This project was a great idea for students to become more engaged in their work and them being incentivised to do more extracurricular activities. Adding the ability to a marketplace was a great way for students to buy things they never could have in a LMS like Bblearn. However, the team did not accomplish everything that would entail a pure "gaming" experience for a course and there are some details that could be added to the project to keep students engaged. Adding more things to the website such as better styling would be a start. The desktop application also does have many opportunities for coders to elaborate on the ideas that we set in place.

7.1 Website Application

The website applications visual presentation could use some work.This could be done by using more advanced technologies such as React or Angular. Also, adding more things in the marketplace for students to buy would improve the project. These items or things to do could be adding a lottery for students to win coins, adding the ability to gift coins to other students, or buying things like extra questions to better the students grades. These optional abilities for the students to use would most likely improve student engagement.

7.2 Desktop Application

On the desktop application future work could include adding more abilities to the super admin so they can delete or edit any course a teacher is teaching. Another desirable feature would be a better UI for the teacher to use. The UI could be customized for the teacher to use by editing the shape and color. Also, fixing the smaller bugs located in the main course creation page should be fixed. These bugs do not affect the application but do make it an inconvenience to the user if they were to run into them. Since Unity is a game engine, making anything in the application is possible and the team encourages new ideas to be implemented into it.

8.0 Conclusion

Team Gaming Ed's motivation for doing this project was to better help Professor Terry Baxter's gamified course in Bblearn. Before this project, he had to manually account for each student's transactions and grades which added hours to his work life. We accomplished making his life easier by building a course creator and course that enables students to become better engaged in their coursework by adding abilities from a video game into a college course. The RTX gamification project accomplished:

- Making a offline course creator in a 3D environment
- Making a LMS with gamified aspects

This project was a huge learning experience for all three of us. We were exposed to technologies such as .NET, mySQL, AWS, and the Unity game engine. Our project is a great start to what future classes could be like in the future. Other schools like NAU could implement this idea into their teaching experience and we think that we made a great start to something big. Our team will further the knowledge we have gained from this course into our work experience. We will take our knowledge of these technologies and apply them into our jobs as well. Gaming Ed has learned a lot about working with a small team and accomplishing a large project. We are sure we will take this experience with us into the future.

Glossary

LMS - Learning Management System.
PHP - A general-purpose scripting language geared toward web development. On a web server, the result of the interpreted and executed PHP code – which may be any type of data, such as generated HTML or binary image data – would form the whole or part of an HTTP response.
RDS- Relational Database
UI - User Interface

Appendix A: Development Environment and Toolchain

Hardware

- Any OS should be able to run this product. A decent graphics card may be needed to edit the Unity application.

Toolchain

- **Visual Studio:** Used to edit the website code and run it on local host
  - Frameworks:
    - Microsoft.AspNetCore.App
    - Microsoft.NETCore.App
  - Packages:
    - Blazored.LocalStorage (4.2.0)
    - Dapper (2.0.123)
    - IronPdf.Native.Chrome.Windows (2022.10.9530)
    - MySql.Data (8.0.29)

- **Visual Studio Code:** Used to edit the Unity C# code
  - Frameworks:
    - None
  - Libraries:
    - System.Collections
    - System.Collections.Generic
    - System.IO
    - System.Linq
    - System
    - System.Threading.Tasks
    - System.Threading
    - UnityEngine
    - UnityEngine.UI
    - UnityEngine.Networking
    - UnityEngine.SceneManagement

- **Unity Editor:** Used to edit the Unity application
- **XAMPP:** Used as local server to host PHP code
- **MySQL workbench:** Used to manually edit database/tables and for testing
- **Blender:** Used to make custom 3D text

Setup

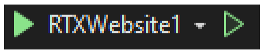
1. To set up Visual Studio packages:
    a. Search “Manage NuGet packages”
    b. Search and install all needed packages

2. To set up Unity Editor
    a. Click “Choose Project”
    b. Choose the whole Unity folder
    c. Click on the project to open it

Production Cycle

To set up Website inside of Visual Studio:

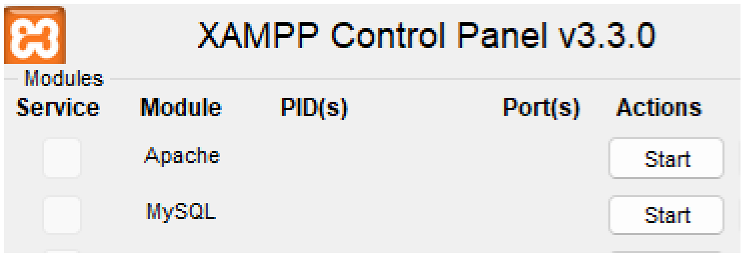
1. Hit the RTXWebsite1 on the top of the page



2. A new window will appear on your computer with the running website

To set up the XAMPP local server:

1. Click start on Apache
2. Click start on MySQL



To set up PHP files:

1. Navigate inside of XAMPP folder
2. Navigate to htdocs
3. Paste UnityApp folder (located in github) inside of htdocs

To build the Unity application:

1. Click File
2. Click Build Settings
3. Click Build
4. Choose which folder to build your project