Immersive Quiz for Spanish Learners

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Austin Klum

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2022

Immersive Quiz for Spanish Learners

By Austin Klum

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Prof. Kasi Periyasamy Examination Committee Chairperson	Date
Prof. Steven Senger Examination Committee Member	Date
Prof. Kenny Hunt Examination Committee Member	Date

Abstract

Austin Klum, J., “Immersive Quiz for Spanish Learners,” Master of Software Engineering, May 2022, (Elliot Forbes, Ph.D.).

This manuscript describes the development of a quiz creation tool combined with a virtual reality component to provide an immersive quiz taking experience for Spanish learners. The quizzes also have an orienteering course aspect as well, where each quiz is comprised of multiple timed locations where all questions must be completed correctly before continuing onto the next location.

Acknowledgments

I would like to express my thanks to the Department of Computer Science at the University of Wisconsin–La Crosse for providing the learning materials and computing environment for my project.

Table of Contents

List of Tables

List of Figures

List of Code Listings

Glossary

ASP.NET Core

Lorem ipsum

ASP.NET Core MVC

Lorem ipsum

Unity

Lorem ipsum

Unity XR

Lorem ipsum

C#

Lorem ipsum

LINQ

Lorem ipsum

Entity Framework Core

Lorem ipsum

SQL Server

Lorem ipsum

Transparent Data Encryption (TDE)

Lorem ipsum

Basic Authentication

Lorem ipsum

JSON

Lorem ipsum

HTTPS

Lorem ipsum

REST

Lorem ipsum

CSS

Lorem ipsum

HTML

Lorem ipsum

Bootstrap

Lorem ipsum

Web API

Lorem ipsum

GUID

Lorem ipsum

Git

Lorem ipsum

Dependency Injection

Lorem ipsum

Razor View Engine

Lorem ipsum

Visual Studio

Lorem ipsum

Unity Editor

Lorem ipsum

Cross Side Scripting (XSS)

Lorem ipsum

SQL Injection

Lorem ipsum

xUnit

Lorem ipsum

1. Introduction

1.1. Overview

The rise of globalism has prompted people of different cultures to increasingly work together and interact with one another. Thus, understanding other cultures and languages will become ever more important. Often times this can be hard to teach, especially in a classroom. Virtual reality can be used as a means to bridge the gap between real-world understanding and classroom knowledge. Virtual reality allows for a more immersive experience. A more immersive experience is a more effective way to engage students and promote learning.

In 2017-2018 there was an initial virtual reality project conducted by Claire Mitchell to take tours of Medellin, Colombia. This project was a success and discussions were made to expand on this initial success. In 2019, there was a grant proposal for development of a new project to further enhance experiential learning. As virtual reality is a vanguard area of development such resources don't exist yet and would require new development. The proposal also requested an orienteering component to be included. Orienteering is an activity where participants "navigate between checkpoints along an unfamiliar course" [1]. The primary purpose of adding an orienteering aspect is to add to the depth of cultural understanding, as orienteering requires the participants to have a more active role in the experience.

The initial virtual reality project was predefined allowing no customizability within the courses. Also, there was no grading aspect of the project. The prior project helped expand students cultural understanding, but there was no built-in grading. The virtual reality wasn't very immersive and required holding a phone in front of the head to look around. The goals of this project want to improve upon these limitations. The project should be customizable, gradeable, and immersive all while garnering student interest and understanding of culture.

2. Requirements

2.1. Overview

The following section is on the requirements of the project and development methodology.

2.2. Development Methodology

The stakeholders for this project is the project owner and sponsor, project advisor, and developer, Claire Mitchell, Elliot Forbes, and Austin Klum respectively. The end users are the students using the virtual reality tool for their learning and the instructors using the quiz management tool for assessment of student's comprehension of classroom material.
The chosen development methodology for a project is an early and influential decision made that alters the course of development. As this was a solo-developer, web, and virtual reality project with a busy project sponsor, the decision was made to make use of an iterative agile methodology. An iterative agile approach focuses on delivering value to the product in fast small increments, rather than all at once. This approach allows software developers to adjust, refine, and review the development process to better provide value and output. This approach also allows for earlier risk identification and the flexibly to easily correct course.

Traditional methodology follows the waterfall approach which "contains five phases of management, where each requires a deliverable from the previous phase to proceed" [2]. The waterfall method is more suit for projects that follow a linear path and is fixed and rigid. The project did not have this rigidity or clarity of output, hence a more Agile approach was taken. The developer also made use of a KanBan board to help keep organized.

2.3. Course Creator

This tool allows the project sponsor and verified users to create orienteering courses for the students to complete. Each course is comprised of locations which has a corresponding photo to accompany the location. To help with immersion and to best utilize the capabilities of virtual reality, the uploaded photos must be 360 Photos, also known as a photo sphere. This type of photo allows for the virtual reality tool to wrap the image around the user making a sphere, such that the user is able to look around as if they were really at that location. The tool also allows for locations to be added, updated, or deleted. Each location has a list of questions. There is no limit to the number of questions per location. The tool allows for each question to be added, updated, or deleted. Each question has a list of answers which can be of one to six possibilities. The tool allows for each answer to be added, updated, or deleted.

Once a course has been completed by a student, a verified user can view the results from the course main dashboard. This dashboard lists the student ID, point score, time score, and total score. As security is important, the database is encrypted and requires an authorized administrator database account to access the data outside of the tool.

For a user to be created, they must create an account with an email and password. Once the account is created, the user is immediately able to login, but will be unable to access any of the course creation or course results pages, limited only to the homepage and user settings. Each verified user has the capability to approve other users. In the user settings page, there exists a link to verify users. The verify users page list all users and their status of verified or not verified. From here a verified user can verify or un-verify other users by selecting the corresponding checkbox and saving.

2.3.1. Users

Figure 1 explains the use case diagram.

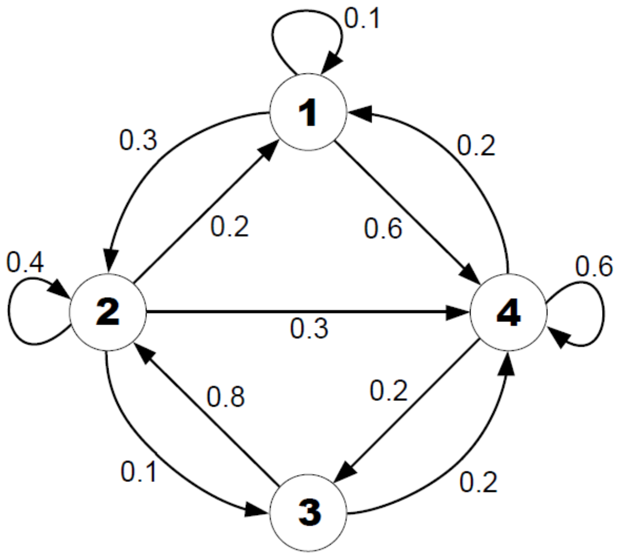


Figure 1. Course Creator Use Case Diagram

2.3.2. Create Course Flow

This section will include photos and written steps on how to use the course creator.

2.4. Virtual Reality Orienteering

This tool is used by the students to complete the created orienteering courses that the project sponsor or verified user(s) created. Upon starting the program the user is prompted to enter their student ID using a "mallet" and tap virtual keyboard. Once the student has entered their student ID, they then select the course via a select list with a point and trigger arrow pointers buttons. After hitting the start button the student begins the course. The first location is loaded up and the student can look around the location. Upon touching the touch pad the first question and corresponding possible answers appears in the world view. As part of the orienteering spirit each question must be answered correctly to continue on to the next question. When a student answers incorrectly the button turns red and is disabled. When answered correctly the button flashes green and the next question and corresponding possible answers load. The student is graded on how quickly they came to the correct answer and, with the orienteering aspect in mind, the time to completely answer all the questions in a location. Each question is worth one point and each location time is worth one point. The score point is awarded divided by the number of attempts. This means that a question with four possible answers the following outcomes are possible: answered correctly has one point awarded, one incorrect attempt has .75 points awarded, two incorrect attempts has .5 points awarded, and three incorrect attempts has .25 points awarded. For the time aspect of the grade, the student has 100 seconds to complete the location, where each second is worth .01 points. This means if the student took 30 seconds to answer the questions and complete the location, they are awarded with .70 points. A time of 45 seconds is awarded .65 points.

Upon answering all the questions for a location, the next location is loaded and the timer is reset. Once all locations have been completed a game over screen appears stating the student's point score, time score, total score, and what the maximum points awarded could be.

2.4.1. Users

Figure 2 explains the use case diagram.

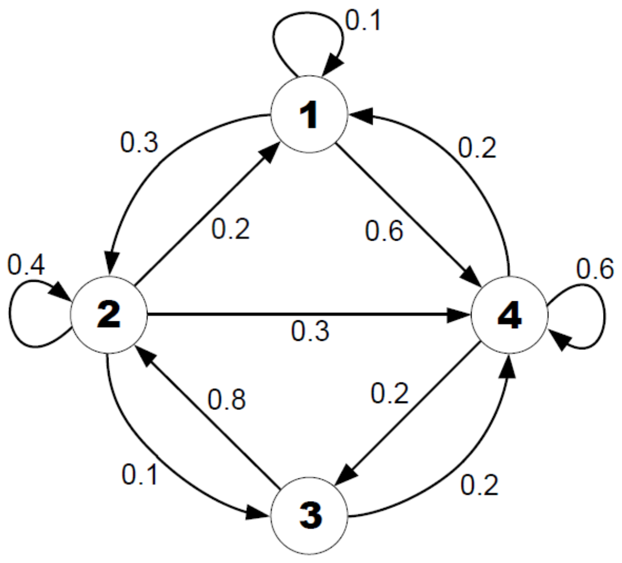


Figure 2. Virtual Reality Orienteering Use Case Diagram

2.4.2. Virtual Reality Orienteering Flow

This section will include photos and written steps on the experience of the student completing a course.

3. Design

3.1. Overview

This section discusses the design of the project including technologies used, classes, database schema, and user interface.

3.2. Technologies

This project uses a combination of ASP.NET Core MVC and Unity XR. The Course Creator was developed with the former and the Virtual Reality Orienteering was developed with the latter.

3.2.1. Course Creator

ASP.NET Core MVC "is a lightweight, open source, highly testable presentation framework optimized for use with ASP.NET Core." [3] ASP.NET Core is the underlying framework that enables development. ASP.NET Core "is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled, Internet-connected apps." [4].

The MVC in ASP.NET Core MVC stands for Model-View-Controller. MVC is an architectural pattern which separates an application into three main components: Models, Views, and Controllers. This separation helps achieve a "separation of concerns", which asserts "that software should be separated based on the kinds of work it performs" [5]. Models represent the data structure, independent of the user interface. Models are responsible for data, logic, and rules of the application. Views represent the user interface and information. Views are responsible for presenting content with minimal logic. Controllers represent the logic and actions for models and views. Controllers are responsible for responding to user input and preforming operations. In summary, models are what it is, views are for what it looks like, and controllers are for how it behaves.

ASP.NET Core MVC was chosen because of the developer's prior experience and the extensive functionalities that the framework provides. ASP.NET Core MVC is open-source and multi-platform, supporting Windows, macOS, and Linux out of the box. ASP.NET Core MVC provides routing which is a useful URL-mapping component. Routing provides for easy link generation without regard to the actual file structure. ASP.NET Core MVC also provides model binding on requests. This makes incoming and outgoing requests easy to process or generate without further processing. Model validation is also built-in using data annotation attributes. These are pre-built or custom attributes within the model that validate on the fly rather than requiring explicit checking. This allows for guaranteeing the state of the model before further processing. Dependency injection is also supported, which is a key feature for building the controllers. Dependency Injection (DI) is a software design pattern, "which is a technique for achieving Inversion of Control (IoC) between classes and their dependencies." [6] A dependency is an object than another object depends on. When a class depends on another class, future changes become problematic. Dependency Injection solves this by using an interface to abstract the dependency, registering the dependency in a service container, and then injects the dependency when needed. ASP.NET Core MVC provides filters which can be placed on controllers so that all actions must meet this filter. Oftentimes filters are added for exception handling or authorization. This way instead of each action requiring authorization, one can require the entire controller with all actions to be authorized. ASP.NET Core MVC is also a great platform for building Web APIs. HTTP content-negotiation with common data formats such as JSON or XML is already supported. The Razor view engine is another key advantage for ASP.NET Core MVC. Razor view engine is a compact and easy template markup language used for defining views with embedded C# code. Razor can be used to dynamically generate web content with a mix of server side and client side code. Tag Helpers are also used with Razor to facilitate creating and rendering HTML. Tag Helpers bind to certain HTML elements and vastly improves their use cases. ASP.NET Core MVC also provides a database object-relational mapper (O/RM) tool called Entity Framework Core. Entity Framework Core provides a "code first" experience to create database entity models by writing code. This cuts down on the boilerplate code required for making database connections. The database can be accessed and queried using LINQ, Language-Integrated Query, a set of technologies based on the integration of query capabilities directly into the C# language. Entity Framework Core makes committing changes to the database simple with automatic change tracking.

3.2.2. Virtual Reality Orienteering

Unity is a cross-platform game engine developed by Unity Technologies with the goal to provide developers with the tools to make game development simple. Unity supports a vast variety of platforms and user experiences. These tools extend to desktop, mobile, console, and virtual reality with support for 2D, 3D, and other experiences. It's also used in other areas outside of game development such as film, engineering, architecture, and automotive modeling. This makes Unity a popular choice, not only for the developer of this project, but for the world at large. "In the fourth quarter of 2021, Unity had, on average, 3.9 billion monthly active end users who consumed content created or operated with its solutions." [7]

The basic components of a game developed in Unity are GameObjects, Assets, Scenes, and Scripts. Every object in a game is a GameObject. Assets are reusable items that can be used throughout the game. They can be of any file type that Unity supports, such as a 3D model, audio file, image. Oftentimes these assets come from outside Unity, created by the developer or other developers which are found in the Asset Store. Scenes contain the objects of the game. Oftentimes these are split into logical groupings such as main menu, individual levels, or the environment. Scripts are what controls the behavior of the GameObjects. Without scripts the game would be static and have no interaction or logic. Unity supports C# natively and is the standard used for scripting. A new script has two functions, `Start()` and `Update()`. `Start()` is called once by Unity before gameplay begins which is used to setup initial configurations. `Update()` is called once per frame update for the GameObject. This is used to handle anything that needs to be done over time in the gameplay, such as movement, triggering actions, or responding to user input.

Listing 1. New Script in Unity

```
using UnityEngine;
using System.Collections;

public class NewBehaviourScript : MonoBehaviour {

  // Use this for initialization
  void Start()
  {

  }

  // Update is called once per frame
  void Update()
  {

  }
}
```

Unity XR Interaction Toolkit is a cross-platform plugin used for virtual, mixed, and augmented reality. XR Interaction Toolkit provides easy built-in functionality to select, grab, throw, rotate objects within a VR scene. This also extends to the UI interactions and the haptic feedback that comes with it. The ability to look around and move within the worldspace is also provided with this plugin. As this project uses a HTC Vive headset, OpenXR was the targeted development platform. OpenXR is an open standard that targets a wide array of virtual reality devices.

3.3. Class Diagrams

3.3.1. Course Creator

3.3.2. Virtual Reality Orienteering

3.4. Database

The database uses Microsoft's SQL Server. SQL Server is the de facto used for .NET projects. SQL Server is a relation database management system which manages and stores information. The standard tool for working with SQL Server is SQL Server Management Studio which makes database changes and transactions easy. Transparent Data Encryption (TDE) encrypts data files at rest. This means that any data files stored are encrypted preventing any malicious attempts to read the database.

3.4.1. Database Schema

3.5. Communication between Programs

This project comprises of two separate programs which must communicate with each other, and do so securely. The Course Creator has a RESTful Web API which requires Basic Authentication to make calls to. Basic Authentication requires a username and password over an HTTP connection. Using just an HTTP connection is not secure though, as the username and password are sent over in plaintext. This means any malicious user can sniff the network and easily obtain the credentials. To counter this the project requires a secure connection using HTTPS, with the 'S' meaning secure. An HTTPS connection encrypts any data sent over it; thereby securing the Basic Authentication credentials. The RESTful Web API provides predefined endpoints for authorized users to use with REST calls. REST stands for REpresentational State Transfer and is a architectural standard for communication on the web. REST is stateless, meaning that the server does not need to know about what state the client is or vice versa. This allows for communication without needing to know the previous messages. A REST request usually is comprised of an HTTP verb, header, URI path, and an optional body containing data. The four basic HTTP verbs are GET, POST, PUT, and DELETE. GET retrieves data or a specific resource. POST creates new data or a new resource. PUT updates data or a specific resource. Finally, DELETE removes data or a specific resource. The header is used for carrying pertinent information about the request being made. This includes the authorization, cookies, caching, and other logistical information for fulfilling the request. The REST requests consume or return data in JSON which is a standard data-interchange format. JSON is easy for humans to read and simple for computers to parse and generate.

4. Implementation

4.1. Overview

This section goes into details on the implementation of the Course Creator and Virtual Reality Orienteering programs. Key examples and general explanations are presented, to help the reader understand the code without rehashing the entire project.

4.2. Course Creator

The Course Creator's primary responsibilities is creating courses and viewing results of those courses. Courses are comprised of locations, questions, and answers; each course has many locations, each location has many questions, and each question has many answers.

4.2.1. Authorization

The Course Creator uses ASP.NET Core Identity to secure the application. This tool makes creating the registration and login pages simple, and has built-in security for securely storing passwords and other personal user data. ASP.NET Core Identity also makes use of user roles to ensure the user is authorized to do certain actions. The verified user role is an example of this in the project. Using a filter on the protected controllers is simple with Identity.

Listing 2. Securing Controllers using Filter on User Role

```
using Microsoft.AspNetCore.Authorization;

[Authorize(Roles = "Verified")]
public class HomeController : Controller
{
  ...
}
```

4.2.2. Courses

The course is the high level object with which the locations, questions, and answers become associated with. A course can be added, updated, or deleted. The `CoursesController` handles all of these actions. Each of these actions has a corresponding View Razor page. The basic course model looks like the following:

Listing 3. Course Model

```
public class Course
{
  public int CourseId { get; set; }

  [DisplayName("Name")]
  public string Name { get; set; }

}
```

The create and edit pages are similar to the other pages for the locations, questions, and answers, so this section will not go in depth on them, but instead elaborate further on the details and delete pages. The details page displays not only the course model information, but also all information relating to the course. To make the related course information modularize and reusable, the developer made use of “View Components”. View Components render a chunk of HTML output within another markup’s file. This breaks up large markup files into smaller parts, reduces duplication of markup content, and provides an opportunity to use logic to control the rendered HTML. Each of the dependent classes on a course has a corresponding View Component which lists the data relating to the course. This can be seen in Listing 4 of the course details page on line 30:

Listing 4. Course Details Razor Page

```
1  @model ImmersiveQuiz.Models.Course
2
3  @{
4  ViewData["Title"] = "Details";
5  }
6
7  <h1>Details</h1>
8
9  <div>
10   <h4>Course</h4>
11   <hr />
12   <dl class="row">
13     <dt class="col-sm-2">
14       @Html.DisplayNameFor(model => model.Name)
15     </dt>
16     <dd class="col-sm-10">
17       @Html.DisplayFor(model => model.Name)
18     </dd>
19   </dl>
20 </div>
21 <div class="mb-4">
22   <a class="badge large-badge bg-secondary text-light" asp-action="
       Index">Back to List</a> |
23   <a class="badge large-badge bg-success text-light" asp-controller=
       "Scores" asp-action="Index" asp-route-id="@Model.CourseId">
       Scores</a>
24 </div>
25 <div>
26   <a class="badge large-badge bg-success text-light" asp-action="
       Create" asp-controller="Locations" asp-route-id="
       @Model.CourseId">Add Location</a> |
27   <a class="badge large-badge bg-primary text-light" asp-action="
       Edit" asp-route-id="@Model.CourseId">Edit</a>
28   <a class="badge large-badge bg-danger text-light" asp-action="
       Delete" asp-route-id="@Model.CourseId">Delete</a>
29 </div>
```

```
@await Component.InvokeAsync("LocationList", new { CourseId =
    Model.CourseId.ToString(), search = "" })
```

Another area of interest for the Courses is the delete action. A course could be deleted independent of the dependencies which destroys the concept of Referential Integrity, which is "a database concept that is used to build and maintain logical relationships between tables to avoid logical corruption of data." [8] When data is deleted independently, the dependencies now have a reference to data that no longer exists. This corrupts the reliability of the data. To counter this the developer made use of cascading deletes. That is to say, when a course is deleted, the delete cascades onto the locations, questions, and answers dependent on that course. The cascading delete can be seen in Listing 5 using LINQ, Entity Framework Core, and Dependency Injection.

Listing 5. Course Cascade Delete

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
{
  var course = await _courseContext.Course.FindAsync(id);

  var locations = _locationContext.Location.Where(l => l.CourseId ==
      course.CourseId);

  foreach (var location in locations)
  {
    var questionsToLocations = _questionContext.Question.Where(q => q
        .LocationId == location.LocationId);
    foreach (var question in questionsToLocations)
    {
      var answersToQuestion = _answerContext.Answer.Where(ans => ans.
          QuestionId == question.QuestionId);
      _answerContext.Answer.RemoveRange(answersToQuestion);

      _questionContext.Remove(question);
    }
    _locationContext.Location.Remove(location);
  }

  _courseContext.Course.Remove(course);

  await _answerContext.SaveChangesAsync();
  await _questionContext.SaveChangesAsync();
  await _locationContext.SaveChangesAsync();
  await _courseContext.SaveChangesAsync();

  return RedirectToAction(nameof(Index));
}
```

4.2.3. Locations

The Location is the container for the questions and answers. The Location is also responsible for the management of the 360 photos. While it is possible to store images and files directly into the database using BLOBS, this is inefficient in storage and retrieval. Instead the developer made use of File System storage. Thus, the location's database record instead contains a reference to where the image lives, rather than the image itself. A location can be added, updated, or deleted. The `LocationsController` handles all of these actions. Each of these actions has a corresponding View Razor page. The Location model is shown in Listing 6.

Listing 6. Location Model

```
public class Location
{
  public int LocationId { get; set; }

  public string Name { get; set; }

  public Guid ImageGuid { get; set; }

  public string ImageExtension { get; set; }

  public string ImagePath {
    get
    {
      return $"/images/{ImageGuid}{ImageExtension}";
    }
  }

  public int CourseId { get; set; }

  [NotMapped]
  public Course Course { get; set; }
}
```

The Location keeps track of the `ImageGuid` and `ImageExtenstion` to create the `ImagePath`. Each uploaded image is given a globally unique identifier (GUID) which "is a 128-bit number created by the Windows operating system or another Windows application to uniquely identify specific components, hardware, software, files, user accounts, database entries and other items." [9]. Assigning each image with a GUID guarantees unique image file names when storing in the File System storage and discourages malicious users from scrapping the File System with predictive names. The process for uploading a new image can be seen in Listing 7.

Listing 7. Uploading an Image

```
private Guid UploadImage(IFormFile image)
{
  Guid imageGuid = Guid.NewGuid();
  string filePath = Path.Combine(Path.Combine(_webHostEnvironment.
      WebRootPath, "images"), imageGuid.ToString()) + Path.GetExtension
      (image.FileName);

  using var fileStream = new FileStream(filePath, FileMode.Create);
  image.CopyTo(fileStream);

  return imageGuid;
}
```

4.2.4. Questions

The Question is the container for answers and contains the content for question being asked. A question can be added, updated, or deleted. The `QuestionsController` handles all of these actions. Each of these actions has a corresponding View Razor page. The Question model is shown in Listing 8.

Listing 8. Question Model

```
public class Question
{
    public int QuestionId { get; set; }

    [DisplayName("Question")]
    public string Content { get; set; }

    public int LocationId { get; set; }

    [NotMapped]
    public Location Location { get; set; }

    [NotMapped]
    public List<Answer> Answers { get; set; }
}
```

4.2.5. Answers

The Answer is the lowest-level object for the Course Creator, containing nothing but the answer content and a boolean for a correct answer. A answer can be added, updated, or deleted. The `AnswersController` handles all of these actions. Each of these actions has a corresponding View Razor page. The Answer model is shown in Listing 9.

Listing 9. Answer Model

```
public class Answer
{
    public int AnswerId { get; set; }

```

```
  [DisplayName("Answer")]
  public string Content { get; set; }

  public int QuestionId { get; set; }

  [DisplayName("Correct Answer")]
  public bool IsCorrect { get; set; }

  [NotMapped]
  public Question Question { get; set; }
}
```

4.2.6. Course Results

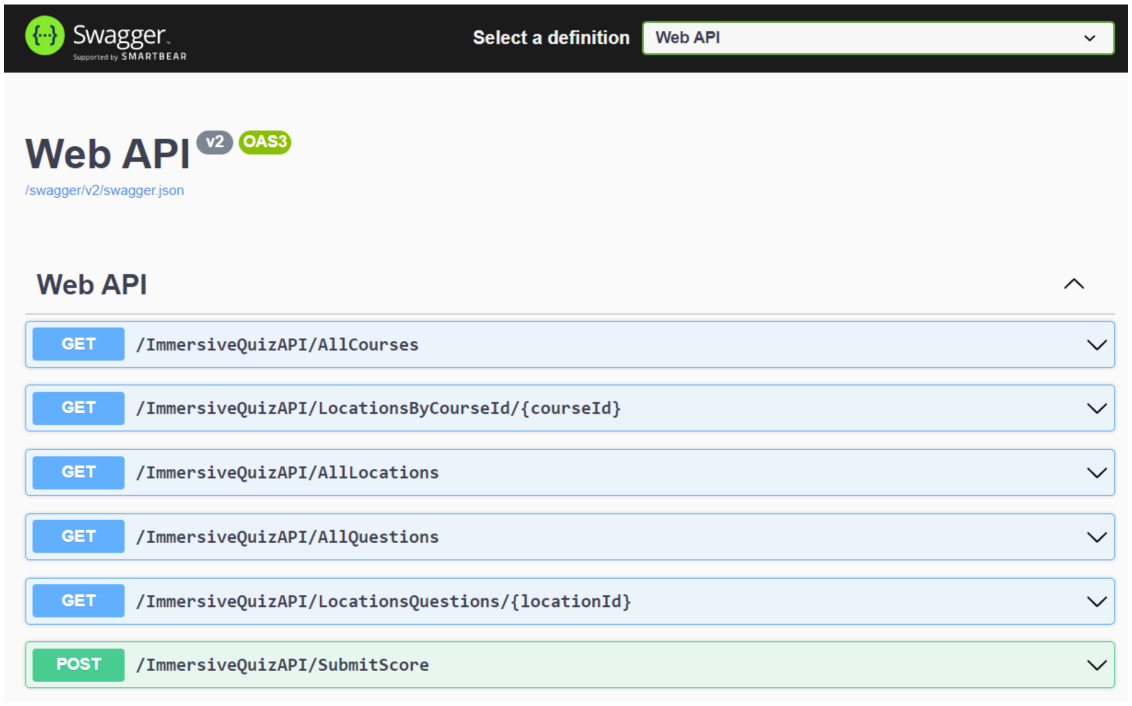
The course results is handled by the Score class which contains the student ID, Time Score, and Point Score. The Total Score is calculated by read only property which adds the time score and point score together. A score can be added, updated, or deleted to give a verified user full control over the scores. The scores will typically come from the Virtual Reality Orienteering program in a POST request. The `ScoresController` handles all of these actions. Each of these actions has a corresponding View Razor page. The Score model is shown in Listing 10.

Listing 10. Score Model

```
public class Score
{
  public int ScoreId { get; set; }

  [DisplayName("Student ID")]
  public string StudentId { get; set; }

  public int CourseId { get; set; }

  [DisplayName("Time Score")]
  public decimal TimeScore { get; set; }

  [DisplayName("Point Score")]
  public decimal PointScore { get; set; }

  public decimal TotalScore {
    get
    {
      return TimeScore + PointScore;
    }
  }
}
```

4.2.7. Web API

The Web API is a RESTful API which controls all of the allowed external endpoints. This Web API is how the Virtual Reality Orienteering program is able to communicate with the Course Creator. Courses are loaded and scores submitted via requests to this Web API. The Web API is secured using a filter with Basic Authentication and a HTTPS connection. Using a tool called Swagger the request documentation is auto generated, providing an easy visualization of the endpoints, schema, and resources available for the Web API. The endpoints can be seen in Figure 3.



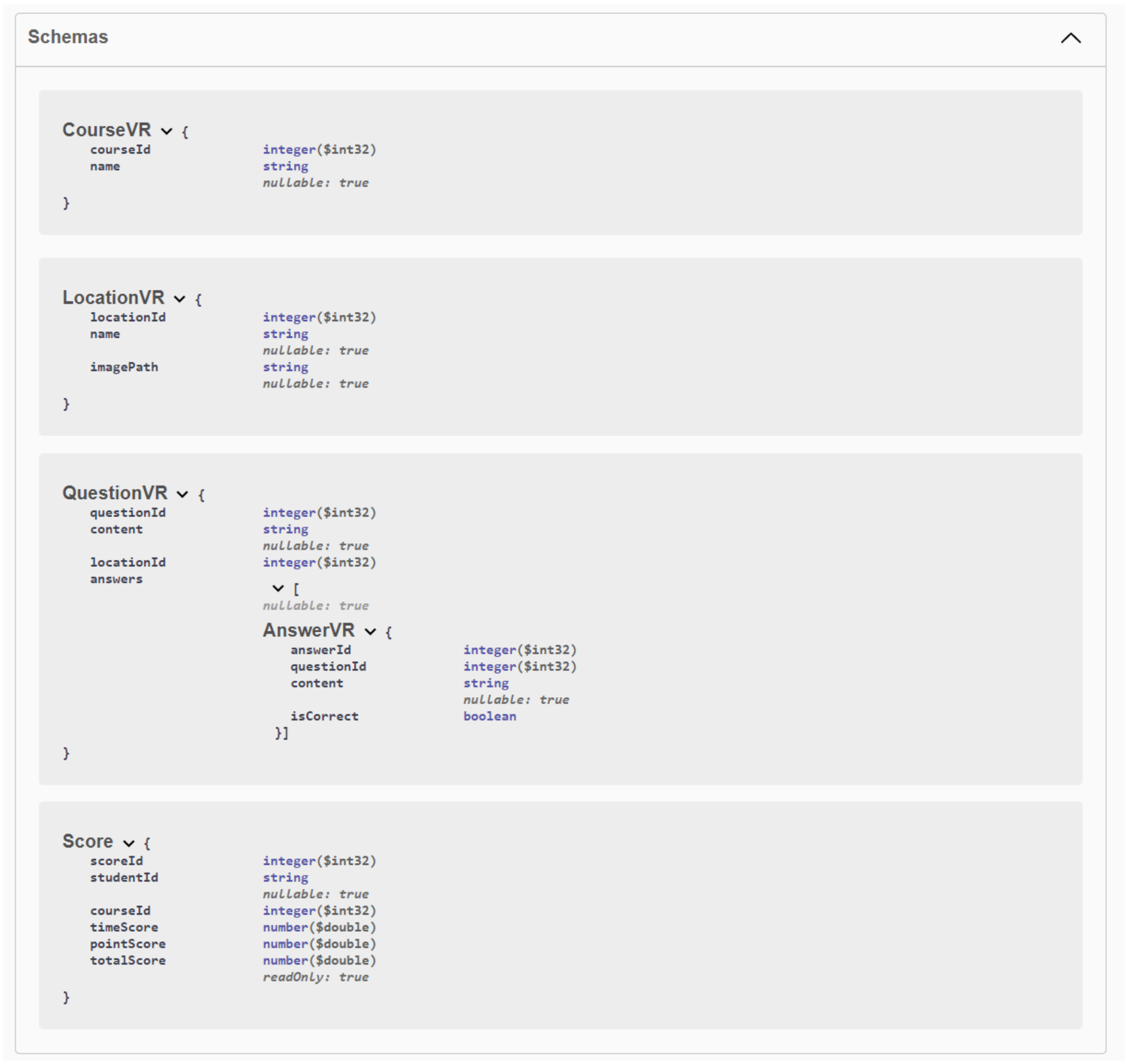
```
GET  /ImmersiveQuizAPI/AllCourses
GET  /ImmersiveQuizAPI/LocationsByCourseId/{courseId}
GET  /ImmersiveQuizAPI/AllLocations
GET  /ImmersiveQuizAPI/AllQuestions
GET  /ImmersiveQuizAPI/LocationsQuestions/{locationId}
POST /ImmersiveQuizAPI/SubmitScore
```

Figure 3. Web API Endpoints

Swagger also auto generates the class schema for the request body's that the endpoints use. The Web API sends only the needed model information and nothing more. Thus, business objects of each of the main components of the Course Creator are used to simplify the requests. These objects can been seen in Figure 4.

4.3. Virtual Reality Orienteering

The Virtual Reality Orienteering primary responsibilities are to display the course and to keep track and submit the results. The Virtual Reality Orienteering program must have authorized communication with the Course Creator program.



```
Schemas

CourseVR {
    courseId            integer($int32)
    name                string
                        nullable: true
}

LocationVR {
    locationId          integer($int32)
    name                string
                        nullable: true
    imagePath           string
                        nullable: true
}

QuestionVR {
    questionId          integer($int32)
    content             string
                        nullable: true
    locationId          integer($int32)
    answers              [
                        nullable: true
                        AnswerVR {
                            answerId            integer($int32)
                            questionId          integer($int32)
                            content             string
                                                nullable: true
                            isCorrect           boolean
                         }]
}

Score {
    scoreId             integer($int32)
    studentId           string
                        nullable: true
    courseId            integer($int32)
    timeScore           number($double)
    pointScore          number($double)
    totalScore          number($double)
                        readOnly: true
}
```

Figure 4. Web API Schema

4.3.1. Authorization

The Virtual Reality Orienteering makes use of Basic Authentication to securely make requests to the Web API in the Course Creator. To access the Virtual Reality Orienteering program the student must be physically present on a computer with a headset, the Course Creator program, and the Virtual Orienteering program. The instructor will proctor the student to ensure the student ID and chosen course is correctly entered.

4.3.2. Displaying a Course

The Virtual Reality Orienteering makes requests to the Web API to get course data. Once a student has entered their student ID, they must select a course. The Virtual Reality Orienteering program calls the endpoint `GET /ImmersiveQuizAPI/AllCourses` to display this list of courses. Once a course is selected, a call is made to the `GET /ImmersiveQuizAPI/LocationsByCourseId/{courseId}` to get all locations for the course. As each location is loaded, another call is made to the `GET /ImmersiveQuizAPI/LocationsQuestions/{locationId}` to load the questions and corresponding answers for each location.

4.3.3. Tracking and Submitting Score Results

In the spirit of orienteering, each location is timed to completion and is added to the total score. A Timer class is associated with the main controller, `VRInputModule`, and provides the functionality for keeping track of the time and returning the time as a score. Listing 11 shows a snippet of the Timer class and the code for the `Update()` which is updated every frame.

Listing 11. Timer Update Snippet

```
using UnityEngine;
using UnityEngine.UI;

public class Timer : MonoBehaviour
{
  public static float startTime = 100;
  public float timeRemaining = startTime;
  public bool timerIsRunning = false;

  ...

  void Update()
  {
    if (timerIsRunning)
    {
      if (timeRemaining > 0)
      {
        timeRemaining -= Time.deltaTime;
      }
      else
      {
        // Time ran out!
        timeRemaining = 0;
        timerIsRunning = false;
      }
    }
  }

  ...

}
```

Throughout the progress of the course, the point score and time score are kept track of in aggregate. When all locations have been completed the UI displays a game over message with the point score, time score, and total score. The Virtual Reality Orienteering program then makes a call to the Web API to `POST /ImmersiveQuizAPI/SumbitScore` with the JSON body of {`studentId: "string", courseId : int, timeScore: float, pointScore: float`}. The `SubmitScore()` method making the call to the Web API is shown in Listing 12.

Listing 12. Sumbit Score

```
private IEnumerator SubmitScore(string studentId, float timeScore,
    float pointScore)
{
  var json = JsonConvert.SerializeObject(new { studentId,
      CourseLibrary.Courses[currentCourse].CourseId, timeScore,
      pointScore });
  var request = new UnityWebRequest($"{WebApi}/ImmersiveQuizAPI/
      SubmitScore", "POST")
  {
   uploadHandler = new UploadHandlerRaw(Encoding.UTF8.GetBytes(json)),
   downloadHandler = new DownloadHandlerBuffer()
  };

  request.SetRequestHeader("Content-Type", "application/json");
  request.SetAuthHeader();

  yield return request.SendWebRequest();
}
```

5. Testing

5.1. Overview

This section describes the testing done to verify and validate that the Course Creator and Virtual Reality Orienteering programs are correct and can handle invalid or malicious input. As part of the iterative design for agile, the programs were tested as new functionality was added.

5.2. Input Validation

Through a variety of means the programs ensure the input entered by users is valid. By preventing invalid input, the programs ensure the data received and presented is correct and avoids issues of unreliable data causing issues downstream. ASP.NET Core MVC provides Tag Helpers which ensures client-side validation takes place. These Tag Helpers use the Data Annotations on the models to determine valid input. A helpful error message is returned to the Razor Page views which describes to the user the invalid input. On the server-side, the Data Annotations on the model are also checked again. This prevents the user from by-passing the client-side and forcefully entered invalid data. When the Data Annotations are not sufficient, the developer manually checked input to ensure valid data. The developer grouped the validation into separate static classes to promote reuseability and modularization.

One common type of attack on web platforms is Cross Side Scripting (XSS). XSS is inserting malicious code into the web page. Once this malicious code runs, the attacker can do anything within the web page to comprise the interactions the victim's has with the page. ASP.NET Core MVC automatically sanitizes input which is the process of disallowing, escaping, or preventing potential code from being executed. Any code entered via inputs is sanitized and cannot execute. Instead the worst case scenario is malicious code is rendered as HTML text on the page.

Another common type of attack is SQL Injection. SQL Injection is inserting malicious SQL statements that will run against the database to either gain information or destroy data. Dynamically generated SQL has more avenues for injection of malicious SQL statements. As Entity Framework Core is how the database connection is created and SQL statements are executed, the only potential vectors for SQL Injection are the values asked for. The risk for SQL Injection is greatly mitigated, as these malicious values are prevented client-side, server-side, and, if not prevented by these other measures, sanitized.

5.3. Unit Testing

Unit Testing is writing code to test your code. Unit Testing are automated tests that test the smallest piece of code. By ensuring all the individual parts of the code are correct, the entire program can be tested quickly, easily, and reproducibly. Unit Tests were created using xUnit. "xUnit.net is a free, open source, community-focused unit testing tool for the .NET

Framework." [10]. One guiding principle for creating unit tests is to ensure reproducible, deterministic tests. Unit tests should avoid external dependencies or rely on other sources in order to determine the result. Unit tests are best used for testing business logic, ensuring expected input and output correctly do the data manipulation needed. Sometimes external dependencies cannot be avoid. This problem can be solved by "mocking" the dependency and returning a predetermined result. In xUnit the common tool to do this mocking is called, Moq. Moq is a popular lightweight mocking library. Moq also supports LINQ to succinctly mock up dependencies. Unit tests are usually of the following format: Arrange, Act, and Assert. Arrange means setting up the test and mocking any dependencies, Act means executing the code piece that's being tested, and Assert means the output of the code matches the expected output. An example of a unit test from the project can be seen in Listing 13 using xUnit and Moq.

Listing 13. Example Unit Test

```
public class WebApiTests
{
  private readonly ImmersiveQuizAPI _webApi;
  private readonly Mock<CourseContext> _mockCourseContext;
  private readonly Mock<LocationContext> _mockLocationContext;
  private readonly Mock<QuestionContext> _mockQuestionContext;
  private readonly Mock<AnswerContext> _mockAnswerContext;
  private readonly Mock<ScoreContext> _mockScoreContext;
  private readonly Mock<DbContextOptionsBuilder> _mockOptions;

  public WebApiTests()
  {
    _mockOptions = new Mock<DbContextOptionsBuilder>();
    _mockCourseContext = new Mock<CourseContext>(_mockOptions.Object);
    _mockLocationContext = new Mock<LocationContext>(_mockOptions.
       Object);
    _mockQuestionContext = new Mock<QuestionContext>(_mockOptions.
       Object);
    _mockAnswerContext = new Mock<AnswerContext>(_mockOptions.Object);
    _mockScoreContext = new Mock<ScoreContext>(_mockOptions.Object);

    _webApi = new ImmersiveQuizAPI(_mockCourseContext.Object,
       _mockLocationContext.Object, _mockQuestionContext.Object,
       _mockAnswerContext.Object, _mockScoreContext.Object);
  }

  [Fact]
  public async Task Post_InvalidScore_ReturnsBadRequest()
  {
    // Arrange
    Score score = new Score()
    {
      StudentId = "1234",
      CourseId = 1,
      TimeScore = -1,
      PointScore = -1
    };

    // Act
    var response = await _webApi.SubmitScore(score);

    // Assert
    var badRequestResult = Assert.IsType<BadRequestObjectResult>(
        response);
    Assert.IsType<string>(badRequestResult.Value);
  }

  ...

}
```

5.4. Acceptance Testing

Acceptance Testing is testing to make sure the programs can preform the functionality that is required. Throughout the iterative agile process, functionality was tested. This means testing the programs as a user would, creating accounts, verifying users, creating courses with location, answers, and questions, etc. Along with this invalid input was tested to ensure a proper error message would appear. As functionality was tested as it was developed, bugs and defect were caught early and often. This reduced the amount of time needed testing everything at the end of the project.

5.5. Integration Testing

Integration Testing is a type of testing that ensure individual components work correctly together as a whole unit. This was primarily seen in the integration and communication of the Course Creator and Virtual Reality Orienteering programs. Ensuring that the Virtual Reality Orienteering program can interface with the Web API, and that the Web API could properly handle these requests, was vital. Without the testing of this area, the project could not function correctly.

6. Conclusion

6.1. Overview

This project succeeds in achieving the goal of creating a customizable immersive cultural virtual reality orienteering program. The two programs working in conjunction provide all the functional requirements expected. In addition, the non-functional requirements of being open-sourced, secure, portable, efficient, and easy-to-use have also been kept in mind during development.

6.2. Challenges

There were numerous challenges throughout the project. The biggest was the time and effort the developer had available. The developer also worked full-time as a Software Developer II at Kwik Trip. While at times related work as a software developer helped the development of the project, most often the work was unrelated and was a hindrance in terms of time and energy. This set the initial expect completion of Spring 2021, to almost a year and half later, with the actual completion during Summer 2022.

Another challenge was the decreased involvement over time of the project sponsor. The project sponsor had a busy workload with other priorities and could not meet often past the initial requirements gathering and early demonstrations. This proved to be challenging as the primary stakeholder's feedback was not taken into consideration as often as an agile process dictates.

As the developer was familiar with ASP.NET Core MVC from working experience, the Course Creator program didn't encounter too many hard blockers. The Virtual Reality Orienteering program on the other hand had many hard stops. The developer had no prior experience working with Unity or designing a VR program. The program flow in a game setting is very different than that of a website. The developer had to learn not only how to develop in Unity, but also understand the order of events and processes that enable real-time interaction within the worldspace. In addition, VR development is still not nearly as standardized or mature as web development. Several techniques or solutions tried proved to be ineffective or considered deprecated. The UI interaction paired with looking around the worldspace took multiple attempts to get right and countless hours to correctly execute. At one point, the developer posted an online listing to pay for a consultant to help solve this issue. Luckily, the developer was able to figure out a solution without the help from a third party.

6.3. Future Work

While the Course Creator and Virtual Reality Orienteering programs achieve the goals for this project, there are multiple areas that could be improved upon, extended, or enhanced. Currently, the programs are portable in the sense they are lightweight and simple to setup. One area of improvement would be to setup the Course Creator to a dedicated server that would be accessible on the UWL campus intranet. The Course Creator was developed as a web application, yet only exists on machines with the program executable downloaded. This would allow courses to persist outside of a local database and allow verified users to view, create, update, and delete data anywhere with a secured connection to UWL. Depending on how the server(s) are setup this would increase redundancy which would provide a more reliable and consistent experience. Database backups and contingencies could be enacted further protecting any loss of data.

The Virtual Reality Orienteering program would also do well for rewriting and increasing modularization. Robert "Uncle Bob" Martin is a famous software engineer and coding author. Five of Martin's principle's have become known as SOLID programming practices. SOLID stands for: Single-responsibility principle, Open-closed principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion principle. Over time the `VRInputModule` grew in size and became a bit of an anti-pattern known as the "God Class". God classes break the 'S' in SOLID programming practices which states "Classes should have one responsibility — one reason to change." [11] During development early mistakes were made and kluged together over time. To reapproach and refactor the program to be "cleaner" and more maintainable would be great future work.

An article [12]
A book [13]
A series [14]
Someone's thesis [15]
Some technical report [16]
A collection [17]
Visited website [18]
Accepted for publication [19]
Submitted for publication [20]
Not published [21]
Conversation [22]

Bibliography

[1] "Orienteering," [Online]. Available: https://www.merriam-webster.com/dictionary/orienteering/.

[2] L. Hoory, "What is waterfall methodology? here's how it can help your project management strategy," 25-Mar-2022. [Online]. Available: https://www.forbes.com/advisor/business/what-is-waterfall-methodology/.

[3] "Overview of asp.net core mvc," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-6.0.

[4] "Overview of asp.net core," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0.

[5] "Common design principles," [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#separation-of-concerns.

[6] "Dependency injection in asp.net core," [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-6.0.

[7] "Gaming poised to continue accelerated growth according to unity gaming report 2022," [Online]. Available: https://unity.com/our-company/newsroom/gaming-poised-continue-accelerated-growth-according-unity-gaming-report-2022.

[8] "Sql joining through referential integrity," [Online]. Available: https://www.w3resource.com/sql/joins/joining-tables-through-referential-integrity.php.

[9] "Globally unique identifier (guid)," [Online]. Available: https://www.techopedia.com/definition/1208/globally-unique-identifier-guid.

[10] "About xunit.net," [Online]. Available: https://xunit.net/.

[11] B. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. Prentice Hall, 2008.

[12] A. B. Cummings, D. Eftekhary, and F. G. House, "The accurate determination of college students' coefficients of friction," *Journal of Sketchy Physics*, vol. 13, no. 2, pp. 46–129, 2003.

[13] I. J. Kuss, *On the Importance of Kissing Up to Your Boss*, 5th ed. Cambridge MA: Dilbert Books, 1995.

[14] L. M. Napster, *Mathematical Theory of Efficient Piracy*, ser. Lecture Notes in Mathematics. New York NY: Springer Verlag, 1998, vol. 3204.

[15] O. P. Qwerty, "History of the goofy layout of keyboards," Ph.D. dissertation, Podunk IN, 1996.

[16] R. Swearingen, "Morpholoty and syntax of british sailors' english," New York NY, Tech. Rep., 1985.

[17] T. Upsilon, "Obscure greek letters and their meanings in mathematics and the sciences," in *Proceedings of the seventh international trivia conference*, V. W. Xavier, Ed. Philadelphia PA: Last Resort Publishers, 1987, pp. 129–158.

[18] J. Tetazoo, "A brief guide to recreational pyromania," Available at http://www.blowinglotsofweirdstuffup.com/guide.html (2005/06/12).

[19] J. Mentor, "Behavior of small animals on fire," (in press).

[20] ——, "Behavior of small animals on fire," 2012, unpublished Manuscript.

[21] ——, "Behavior of small animals on fire," 2012, unpublished Manuscript.

[22] S. Freud, Personal conversation, July 2012.