March 2019

Post Lecture Review

Lucas Ruiz Lebrao

Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Repository Citation

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact digitalwpi@wpi.edu.
Post Lecture Review (PLR)
A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree in Bachelor of Science
In
Computer Science

SUBMITTED BY:
Lucas Ruiz Lebrao, Computer Science

ADVISED BY:
Michael Ciaraldi, Computer Science
Abstract

Many different systems are available to seek student evaluation of a course. However, this evaluation is typically only done at the end of the course; this limits its usefulness. This project develops the concept of getting student feedback throughout the course, in order to assist students in providing earlier feedback to the professor. This prototype is integrated with Canvas, so the system has access to already existing course information, as well as the means to send questionnaires.
Acknowledgments

I would like to thank my advisor Michael Ciaraldi for accept my project proposal, providing valuable input and insight throughout this project. Professor Ciaraldi was quite supportive during our meetings, both in person at WPI and online through Zoom when I was unable to continue studying within the USA.

Additionally, my cousin Vinicius Gomes, who’s greatly knowledgeable on Linux and Server Hosting, taught me many different things to have PLR’s Linux Server up and running. He also provided me a great deal of support in figuring out the many subtleties and strange behaviors and assistance with related server issues.

Also, I have many thanks to all my friends that took their time to be a sounding board to different ideas implemented in this project and sometimes review my English.

Lastly, without the great patience and support that my family provided, I would not have been able to finalize this project or much less reach so far.
# Table of Contents

Abstract................................................................................................................................. i

Acknowledgments.................................................................................................................. ii

1. Introduction ........................................................................................................................... 1
   1.1. About ............................................................................................................................... 1
   1.2. How to use PLR .............................................................................................................. 1

2. Background ........................................................................................................................... 2
   2.1. Existing Methods and Tools .......................................................................................... 2
       2.1.1. College Student Evaluation Forms ........................................................................ 2
       2.1.2. Campus Labs ......................................................................................................... 3
       2.1.3. DropThought ......................................................................................................... 4
       2.1.4. Qualtrics ............................................................................................................... 4
   2.2. Current Problems ........................................................................................................... 6

3. Technology ............................................................................................................................ 7
   3.1. Reasoning ...................................................................................................................... 7
       3.1.1. Platform .................................................................................................................. 7
   3.2. Tools .............................................................................................................................. 7

4. Architecture ......................................................................................................................... 9
   4.1. Model View Controller ................................................................................................. 9
   4.2. Canvas ........................................................................................................................... 10
       4.2.1. Learning Interoperability Tools (LTI) ................................................................. 10
       4.2.2. Canvas REST API .............................................................................................. 11
   4.3. Additional Tools .......................................................................................................... 11
       4.3.1. Application Server .............................................................................................. 11
       4.3.2. Programming Environment ................................................................................. 11
   4.4. Disambiguation ............................................................................................................. 12
       4.4.1. User Interaction with Canvas and the PLR Application ........................................ 12
       4.4.2. PLR Database and Canvas Database ................................................................... 12
   4.5. Diagrams ...................................................................................................................... 13
       4.5.1. Network Architecture ......................................................................................... 13
       4.5.2. Database .............................................................................................................. 14

5. Approach ............................................................................................................................... 15
   5.1. Setup ............................................................................................................................. 15

pg. iii
5.2. Back End Development ............................................................................................................. 15
  5.2.1. Models .................................................................................................................................... 15
  5.2.2. Database ................................................................................................................................. 17
  5.2.3. Controllers............................................................................................................................... 20
  5.2.4. Editor Templates and Model Binding ....................................................................................... 20
5.3. Google Apps and Questionnaires .................................................................................................. 22
5.4. Questionnaire Data ........................................................................................................................ 24
5.5. Google Issues ................................................................................................................................ 24
  5.5.1. Limitations................................................................................................................................. 24
  5.5.2. Re-Evaluation of Thinking ........................................................................................................ 24
5.6. Self-Designed Forms ...................................................................................................................... 25
5.7. Questionnaire Data II .................................................................................................................... 25
  5.7.1. Statistics Generation .................................................................................................................. 25
  5.7.2. Presentation .............................................................................................................................. 25
5.8. Server Hosting ............................................................................................................................... 26
  5.8.1. The LAMA Stack ....................................................................................................................... 26
5.9. Canvas Integration ......................................................................................................................... 27
  5.9.1. Learning Tools Interoperability ............................................................................................... 27
  5.9.2. REST API and its usage ............................................................................................................ 29
6. Analysis ............................................................................................................................................. 30
  6.1. Prototype Results ......................................................................................................................... 30
  6.2. ASP.NET Core as a Tool ............................................................................................................... 31
  6.3. Versatility of Canvas APIs .......................................................................................................... 32
  6.4. Google Limitations as a Tool for System Development ............................................................... 33
  6.5. Possible Improvements as a Tool for System Development ....................................................... 35
7. Conclusion ......................................................................................................................................... 36
8. Glossary .............................................................................................................................................. 37
9. Appendix .......................................................................................................................................... 40
  9.1. Server Configuration .................................................................................................................. 40
    9.1.1. Utilized Software Versions ..................................................................................................... 40
    9.1.2. ASP.NET Installation ............................................................................................................ 41
    9.1.3. MySQL Installation ............................................................................................................... 41
    9.1.4. Apache Server ....................................................................................................................... 43
9.1.5. Supervisor Installation and Configuration ................................................................. 46
9.2. Canvas App Setup ............................................................................................................. 47
  9.2.1. XML Config Builder .................................................................................................. 47
  9.2.2. Canvas Configuration ............................................................................................... 48
  9.2.3. Canvas REST API ..................................................................................................... 50
  9.2.4. XML File .................................................................................................................. 53
10. Attachments ..................................................................................................................... 54
1. Introduction

1.1. About

This project, Post Lecture Review (PLR), provides professors and students with the means to quickly solicit and provide feedback on specific lectures taught by the professor, by completing simple evaluation questionnaires.

This tool is developed as an App built for Canvas, a learning management system from Instructure that provides professors a central location containing the whole coursework, assignments and communication with students. Canvas provides tools that allows remote applications to access and interact with the available information, through its REST API and App Development.

1.2. How to use PLR

Within the opened screen, the Professor must register all Lectures; including their Titles, Date and Time, Duration, Topics and their Sub-Topics, and any Additional Questions that are deemed necessary.

After PLR has all the Lecture information, it will then send Questionnaires to existing Students within the Course after each Lecture is completed. Students fill out aid Questionnaires and provide Lecture Feedback that the Professor can review and adapt. The Professor can change following classes to include poorly reviewed content, provide additional information via Canvas to Students or even organize additional teaching sessions for students with higher difficulties.
2. Background

Given there is a broad spectrum of approaches on our topic, Course Feedback, there is no formal history of how feedback for teaching courses were done over time. Each school/college have developed their own methods or imitated others over time, branching into their own methods. Thus, I’ll describe a few current approaches I could study and access. The following are all currently used methods to assist professors in receiving feedback from students.

2.1. Existing Methods and Tools

2.1.1. College Student Evaluation Forms

Most commonly, Colleges have end of course evaluations done through a standardized form, which provides general questions with a rating each, on both the Professor and the Course itself. For example, WPI has the following format for Course Evaluations:

![Figure 1 - WPI Course Evaluation Form](image)

Many other colleges take a similar approach, with many examples of both physical and digital evaluation forms. Usually they follow similar guidelines:

- Wide array of questions
- Scale based grading
- Grade Expectations
- Questions on Subject Taught
- Questions on Professor Himself
- Broad Feedback Questions
However, despite the wide array of questions and ways of providing feedback, these Questionnaires are rather ineffective. This is due to a few issues surrounding it, primarily due to it being done at the end of the Course (where it doesn’t affect students anymore) and often being anonymous, which leads to the following problems:

1. Polarized Feedback: During feedback only students that either hold strong positive or negative opinions will fully consider giving proper feedback. Students that have neither positive or negative impressions will provide random or generic feedback, as the format allows for any responses to sound valid.

2. Biased Opinions: People are more prone to exaggerate opinions given the Anonymous nature of feedback, hence even their feedback isn’t accurate.

3. Lack of Motivation: Since the students already completed the course, most of them have no real reason to spend their time properly answering the course feedback. It is most likely that they will simply randomly answer ratings and provide meaningless comments when asked to write.

4. Employment depends on Quality of Feedback: In many colleges, Professor Evaluations affect their employment status, which leads to Professors shifting their teaching methods and course difficulty to gain more favorable student reviews.

2.1.2. Campus Labs

Different from standard Feedback Tools, Campus Labs provides a whole management system for a College, with many different Modules that can be implemented to fit your College’s management needs. Despite the fact of this tool also being capable of providing the means for both Student Course Evaluations and its Analysis, CampusLabs real intent is on the improvement of a college organization.

The most interesting aspect of this tool is about its focus in analyzing large amounts of data, by correlating grades, questionnaires, colleges, department and professors, which results in far more data than standard Course Evaluations, providing much more interesting data.
2.1.3. DropThought

Dropthought is possibly the application with the most similarities to Post Lecture Review, where it follows the simplistic approach of providing simple forms to be completed by students to provide the professor with feedback over a Course.

It features simple forms, with a textbox area for comments on said topic and a rating for the overall experience, ranging from Excellent to Poor in 4 different ratings. Additionally, it also follows the Learning Managing Systems frameworks, capable to send those forms within Canvas and similar systems. Lastly, it features its own mobile app, where course instructors can easily read feedback received.

However, its format is less flexible than PLR, where each form can only provide space for a single individual topic and its rating. This makes it difficult to receive more precise details on what exactly students struggled with. Additionally, Dropthought promotes its anonymity of student responses, as its feature, where student names would not be displayed with their responses and ratings. But it limits how well the professor can understand the student troubles, where he’s able to determine how well this student is doing throughout the whole course, and then providing more accurate assistance.

2.1.4. Qualtrics

Unlike the goal of other tools, CampusLabs or DropThought, to provide feedback for the improvement of Study Environments, Qualtrics focuses in generating more accurate data for Companies.

Qualtrics describes its service on the way it handles and describes data, by determine of what led them to finding such results. To do so, they first explain how Qualtrics collects information, having their information dubbed as the Objective Data (O-Data). This refers to all collected results, without giving much context of why they reached such results, either good or bad. Hence Qualtrics desires to understand the patterns and inputs that led the client find the resulting O-Data. To do so, Qualtrics compiles all user experience into their so-called Experience Data (X-Data), which algorithmically determines what common factors give certain results. This
helps the client to determine what actions were beneficial for his product, hence finding actual information on how to improve and correct business decisions.

Qualtrics divides its experience analysis in 4 different products:
- Qualtrics’ Customer Experience™
- Qualtrics’ Employee Experience™
- Qualtrics’ Brand Experience™
- Qualtrics’ Product Experience™

Despite Qualtrics not having the same objective as this project’s, it is quite interesting to analyze given its methods of quick positive/negative feedback which phone/web notifications, which only requires a few clicks. Hence it becomes able to correlate both sales data with customer/employee satisfaction during the process to gauge how well its business practices are going.
2.2. Current Problems

Currently there is much debate on the actual worth/effectiveness of Student Evaluations, where there are great concerns that the great majority of feedback is biased and non-informative, and professors end up shifting their teaching methods to improve student reviews (if their institution uses student evaluations to determine contract renewals). Another problem is the “online disinhibition effect”, where student reviews are usually anonymous and such feedback only ends up enabling unethical and harsh evaluations in a non-constructive way. That is assuming students complete the Course Evaluations at all; since they usually are not obligatory and do not affect the student themselves. Since the courses have already completed by the time of evaluation.

Given those issues, the PLR attempts to reduce such troubles on the way it is implemented and its importance for both students and professors.

First, PLR will remove the anonymity of Questionnaires, where responses are linked to students that gave them for multiple reasons: (1) provide an incentive to students in completing questionnaires by gaining a small amount of credit towards final grade, and (2) allow the professor to directly determine students with greater problems in the Course and possibly address said issues through comments or direct conversation.

Secondly, PLR affects the students directly in their performance in class, since it is in their best interest in clearing out harder topics and questions from Lectures. Therefore, the feedback of students will be more precise and relevant to them, as the professor will have the means to assist and clarify confusion.

Whether PLR is factored in Institution evaluations or not, the feedback provided will assist professors in reducing student frustrations and difficulties during the course, which in turn will improve the overall reviews done by students.

Lastly, PLR’s objective is to reduce the effort necessary by students to receive assistance and additional teaching materials, through quick and simple questionnaires which evaluate topics taught in the Lecture, from 1 (No understanding of topic) to 5 (Complete comprehension). Additionally, PLR provides space for more specific comments or questions, where the students can specify additional issues if necessary.
3. Technology

3.1. Reasoning

3.1.1. Platform

PLR’s goal is to provide feedback without being intrusive or requiring additional registration/downloads. Hence the following points are what were kept in mind when determining the implementation of our System:

- A web application to provide both students and professors with the means to interact with our system.
- A database to collect and organize our data.
- A server to run both our application and database.
- A public web address to allow our system to be accessed.

However, if we were to develop our system from the ground up, professors would be required to perform many administrative tasks, such as listing all courses that are being taught and manually populating their courses with students.

We have therefore chosen to create an application that works in conjunction with already existing Learning Management Systems (LMS), such as Canvas. To do so, developers began adopting standards to simplify development of services that work across different LMS’, called Learning Tools Interoperability (LTI), which determines how authentication and communication between Apps and LMS’ work.

Since WPI heavily utilizes Canvas to manage a significant number of its taught courses, PLR was developed following the LTI standards to work with existing courses. All testing has been done within Canvas. There are a few professors that still create their own websites to manage their coursework, but those are prone to bugs, lack of commonly used tools, and more confusing navigation.

3.2. Tools

To create PLR, we determine it required the following things:

1. An LTI Application: An Unix-based application, in this case a .NET application. Canvas will simulate the application within its website.
2. Our own Database, as Canvas does not allow us to create the necessary tables to hold PLR exclusive data.
3. A server to run our application which can be accessed by Canvas.

Documentation for development of Canvas Applications is rather scarce, given both LTI and .NET applications have been heavily modified within a short period of time, with only a few current guides with concise information for development. On the other hand, there was plenty of information in configuring Servers and Databases, which leaves only the application as the most complicated part during development.
And since the most available information explain the usage of ASP.NET applications to interact with Canvas, PLR was developed using such technology. There are positive points about utilizing ASP.NET for development:

- There is a fair amount of documentation on ASP.NET development, as well as tools and guides to assist its development.
- The Model-View-Controller approach to ASP.NET development allows easy configuration of both Database and Web Pages, that can be expanded upon.
- There are many available libraries made by Microsoft, which provided more stability during development as well as consistency for usage.

The only disadvantage of using ASP.NET is that it is developed in C#, which is only natively supported on Microsoft Windows systems. Since most of our experience in running and working with servers was in a Linux environment, we utilized Mono to run our application in Ubuntu 18.04.

Mono is a software platform which provides the means to developers of .NET applications to deploy cross-platform applications in non-Windows environments which do not natively support C#. Additionally, Mono is utilized to provide better development tools to any software systems, such as Android, Linux, macOS, etc.
4. Architecture

Due to Post Lecture Review implementing different frameworks and patterns, the Architecture has some complexity. The system was developed with being a Canvas App in mind, so it followed both the Canvas LTI Standard and the ASP.NET Core MVC framework, since LTI is more commonly implemented with ASP.NET.

4.1. Model View Controller

Hence, the project was created in ASP.NET Core 2.0 MVC, written in C#. The ASP.NET Core architecture with the Model-View-Controller assists with the following in a Project:

- **Model**: Equivalent to Objects in Object Oriented Design, with automatic Database Table generation with the creation/modification of Models. Models can be created either Code First or Table First, and additionally provides the means to finely define and adjust the relations between Tables, as well as different ways to seeding the Database.

- **View**: The ASP.NET implementation of web pages, which uses Razor Pages for its versatility. The ASP.NET architecture provides easy usage of its Models directly within its .chtml pages, reducing the need for additional JavaScript and jQuery code.

- **Controller**: The heart of the MVC architecture, which connects the users to the Web App, and handles all the business logic between Models and Views. The Controllers are quite straightforward to implement and have all routing sorted out, where its routing is based off the project File Structure, without modification of other files to work with the Controller. By default, it follows the following web path:

  \[http://<project_url>/<controller_name>/<controller_method>/<arguments>\]
4.2. Canvas

4.2.1. Learning Interoperability Tools (LTI)

The ASP.NET application communicates directly with Canvas Instructure as an App, which accesses system web address and simulates it within Canvas. The diagram above highlights the basic logic of the LTI pattern, where Canvas (Learning Platform) and PLR (Learning Tool) meet halfway to connect to each other, confirm the connection and then exchange data.

To do so, it is first necessary to generate an XML\(^1\) that must be added to Canvas to connect Canvas with the PLR Server. Then begin to add our LTI hooks to access different sections of Canvas. With the XML set up, we insert it into Canvas at the Course Admin section. Full instructions for the LTI hook-up are detailed on the Appendix 9.2. After adjusting the basic Calls and Responses on our Project, we can communicate with Canvas.

Additionally, it is possible to configure OAuth to match the users accessing the Web App, which requires the administrator’s Consumer Key and Shared Secret. The whole Authentication process can easily be set up using IMS Global’s LTI Library (the authority on LTI).

---

\(^1\) XML Config Builder: [https://www.edu-apps.org/build_xml.html](https://www.edu-apps.org/build_xml.html)
4.2.2. Canvas REST API

All data transmission between Canvas and the App is entirely done via Canvas’ REST API. REST API’s simply consist of GET/PUT/POST/DELETE calls to the service’s API, which are language independent, since they are HTML calls utilized in every system.

Canvas defined multiple different method calls that can provide a variety of Course, Student, Professor and other kinds of information. The whole library of REST API calls can be found at the Canvas LMS – Rest API and Extensions Documentation².

This API allows us to access Canvas data to populate our application with Course and Student information. Additionally, the API will be used to create and direct automated messages to students after the Lecture has terminated.

4.3. Additional Tools

4.3.1. Application Server

The server was hosted within WPI’s VM Network, running on Ubuntu 18.04. Despite the whole project being coded in C# and not being natively supported by Linux environments, we utilized Mono⁴ which provides a C# compiler that compiles cross-platform code to run in non-native environments. Additionally, our Database will be built in MySQL, to handle all application data.

4.3.2. Programming Environment

The PLR MQP Project was entirely developed with Microsoft Visual Studio 2017 Community Edition, with a few additional libraries to provide Authentication, better View visuals (PrimeUI/jQueryUI), and tools for debugging and database testing.

---

² What is a REST API?: https://idratherbewriting.com/learnapidoc/docapis_what_is_a_rest_api.html
³ Canvas LMS – REST API and Extensions Documentation: https://canvas.instructure.com/doc/api/all_resources.html
⁴ Mono-Project – Cross Platform, open source .NET framework https://www.mono-project.com/
All the project’s source code is available in GitHub, linked at Section 10 - Attachments.

4.4. Disambiguation

4.4.1. User Interaction with Canvas and the PLR Application

The original design of PLR’s Architecture originally was going to keep the Students entirely away from PLR’s System, where all Questionnaire links would be from Google. The reasoning behind this is to avoid any external access directly into PLR, since professors themselves reach PLR within Canvas.

However, due to complications in managing the generation of Forms within the Project’s Google Account, this original approach had to be changed to meet delivery deadlines. Additionally, further research on using the Google API revealed some limitations of the current API for Google Apps Scripts (in 2015)\(^5\), with no changes until now (2018).

The current PLR Prototype generates its own forms, with all routing and logic handled by the *FormsController*, responsible for anything related with Form Routing, Response Generation and Storage.

4.4.2. PLR Database and Canvas Database

The Post Lecture Review works off two different Databases: both the PLR’s own Database and Canvas’ Database. Despite being able of storing some data within Canvas using the REST API, we chose to manage our own Database for multiple reasons.

The first is due to the fact PLR does many calls to the Database to both populate, read and modify entries, as well generate statistics from student responses. If we were to use REST to store and retrieve everything PLR utilizes, it would generate far too many calls to Canvas. The excess of additional calls would create issues in larger scale implementation of PLR, as there would be an increased chance of lost and corrupted packages, that would require additional work and processing power to handle.

The second reasoning is because it greatly facilitates development, as generated tables can quickly be tested, created, modified and deleted throughout development. Additionally, having PLR’s own Database provides the means to directly observe the data within and additionally, read its logs.

Lastly, PLR can easily seed its own Database, allowing for faster testing and baseline for debugging, since it always would have the same data kept every iteration.

---

\(^5\) Google Issue Tracker, “Execution API – can’t use service account” Topic: [https://issuetracker.google.com/issues/36763096](https://issuetracker.google.com/issues/36763096)
4.5. Diagrams

4.5.1. Network Architecture

4.5.1.1. First Iteration – Google Forms

Figure 8 – First Iteration, Google creates and handles forms logic

4.5.1.1. Second Iteration – Self-Made Forms

Figure 9 - Second Iteration, PLR creates and handles forms logic
4.5.2. Database

Figure 10 - Post Lecture Review SQL Model Relationship Diagram
5. Approach

The PLR project was an idea that came from interviewing a few previous WPI students about problems they had throughout their studies until Graduation. The most recurring comment was about confusions and difficulties among a few Courses they took, whereas one noted the “pointlessness” of end of Course Reviews.

These conversations inspired the idea of providing faster feedback to allow the Professors to adapt their Courses or supplement materials for topics where the class was studying. Some students struggle to reach Professors for a myriad of reasons, where an easily reachable platform allows Students to pass their own comments or current problems with the Course.

Thus, the Post Lecture Review was created to provide this Platform.

The Project proposal defined the following milestones for the project:

1. Creation of a Prototype of a Teaching system to assist both students and professors via quick feedback and means to determine major areas of student difficulties during the progress of the Course.
2. The usage of Canvas to implement the Prototype as an App, which will use its REST API to pull Course data, including Lectures, Students and Topics.

5.1. Setup

Given the 2nd Proposal of this Project, we began looking into the different ways our system could be implemented as an App within Canvas. The documentation showed that Canvas followed a standard pattern that allowed the easy implementation of applications within any Learning Tools that followed LTI guidelines, with minimal code adaptation to implement within more than one website.

This Pattern is called the Learning Tools Interoperability (LTI), which details how an application can interact with any Learning Tool, with simple hooks to allow system authentication, data read/write access within the Course using applications, as well as additional features that would incorporate the application within the LT in question.

Canvas’ own documentation provided simple enough examples of how to create a LTI application, which followed the ASP.NET framework. However, there was very little information on how to implement this Application in other programming languages, hence C# was the final choice.

The project was built within Visual Studio Community 2017, using its latest stable version of ASP.NET Core 2.0 with the MVC architecture framework.

5.2. Back End Development

5.2.1. Models

With a selected language, platform and framework, we began familiarizing with the ASP.NET framework, beginning to work on the Post Lecture Review models to determine how
our Database would build its table. From the lowest level, we looked at the definition of a Lecture, determining the necessary models to fill our Course with multiple Lectures.

With the Lecture Model defined, we created the Views and Database Schema to test the Model within the web-browser. However, within the browser it wasn’t possible to see and/or add data to the children models with the auto-generated Lectures Controller and its Views. We split the problem into three different parts; (1) generating a view which allowed to create a whole lecture, (2) properly referencing all variables within the page with the corresponding models and (3) defining the relationship between Lecture and Topics/Questions in the Database, so each model could be saved into the Database and still refer to each other. This issue and solution will be further expanded in the following sections.
5.2.2. Database

With the PLR’s Models created, it was time to begin implementing the Controllers that would handle both access and modification of Models. Within Visual Studio, it is quite simple to create a default Controller for a Model with List, Create, View and Edit Views by simply right-clicking the Controllers folder and selecting New Controller. Then we will be prompted to select the target Model and Database.

ASP.NET handles its connection with the Database by utilizing a Context file, where we describe in detail how the Database must be built. This is done through DbSet attributes, one for each Model within the project. All these attributes are set/modified through methods, including the different relationships between each Model, so our Context understands how to build the Database. Additionally, we also provided initial entries for all Tables, so our Database is created with testing data.

The following image displays PLR’s DbContext file. Each Model has a corresponding DbSet, where all relevant information is described within the minimized region TableRelationships and all default entries for each table on the other minimized region TableSeeding.
By including our DbContext within every Controller’s constructor, the Entity Framework (EF) allows all Database calls to be made using methods within each DbSet attribute. There’s no need to handle direct Database queries, as the Entity Framework Core (EF) provides the means for very complex queries in few lines of code.

The following image shows an example of how the Lecture Table is accessed via code within the LecturesController. In this example, we first call the local attribute `_context`, which refers to the DbContext, then access the table _Lectures_ and call all entries within it.
The two Includes within the call of the Lectures Table, means that all connected Topics and Questions will be included with the query, as EF always queries the least amount of information possible from the Database.

However, EF would not be able to connect relevant relationships and dependencies, without having those defined within the DbContext file. This is done at the OnModelCreating() method of shown on Figure 12, where these definitions were previously hidden on the region TableRelationships. Here’s an excerpt of the code relevant for the call in Figure 14:

Lastly, within the other region TableSeeding, we have the default entries for the system’s Tables. The process to insert starting Database entries is simple, where Objects of each Model are directly inserted into their corresponding table. The only potential problem is potentially making incorrect references that violates any table rules, in this case, making non-existing references.
5.2.3. Controllers

Now, with our Database properly set up, we can now return to building our Controller logic. By utilizing the Visual Studio assistant to create a Controller using a specified Model, a Controller with List, Create, View, Edit and Delete was generated, with all the necessary initial routing logic to interact and provide Data to automatically generated Views. All Views were created within the Views folder, at the folder with the corresponding name of the new Controller.

As previously mentioned, this generation does not consider nested Models within the target Model, hence the creation and editing Views only have fields for the attributes. To correct this problem, it was necessary to modify most views to allow the creation of Topics and Questions with a newly made Lecture.

5.2.4. Editor Templates and Model Binding

First, we determined how Razor Views worked, since it provides many special tags and commands to intertwine the Data between both View and Controller. Razor Views allowed us to bind a Model to the relevant View, which allows this page to directly refer to attributes within said Model. Additionally, when a Form is submitted within this View, it will dynamically build the POST message with the correct attribute names and values. The POST message will be received by one of the Controller’s methods, which will bind the message contents into an Object that is one of the passing attributes for the said Method.

However, neither View or Model Binding would be able to properly bind Models with nested Objects to them, since the View only knows how to build its bounded Model. This means the View wouldn’t be able to generate fields to create the nested Models, nor properly write a POST message with their necessary information. Hence the Controller would not be able to make sense of the loose POST information and the Sub-Models related to the parent Model, nor differentiate their information from each other.

To solve these issues, we took a closer look at the default Creation Method of a Lecture Object, where we had the standard Model Binding as the argument for the Create Method call. Model Binding simply associates both input attributes from the View with the matching name attributes of a Lecture Model. So logically, it should be able to receive the Sub-Models (in this case, Topics and Questions) and generate complete Lecture object from the View’s data.

However, a single View can only be bound to a single Model, where it will only handle the attributes for a Lecture. Which is why we utilize @Html tags with the same attribute names to create all input fields of the View, otherwise the page will throw an error. Which is why, there is a need of additional Views to bind the nested Models, a view for Topic and a view for Question. However, even if these new views were to be made partial views to the Lecture view, the resulting POST message would ignore these partial views data.
To solve this issue, ASP.NET provides both View and Editor Templates for Models, where we define what an Editor for a Model requires to have to build the entire Object. This also applies to the parent Model, which can recursively draw all children Editor Views, when provided a @Html tag in which we ask the View to draw the Model’s Editor or View Template. Above there is an example of how the Lecture EditorTemplate was built, where the sub-models are only referred to by their corresponding EditorTemplates. The following image displays how the Question EditorTemplate looks like, having the Topic’s Editor almost the same.
5.3. Google Apps and Questionnaires

While searching for methods to generate Questionnaires via code, I’ve discovered that Google had a large library of utilities that could be utilized in ASP.NET projects through its API Library. Being familiar with some of the commonly used Google products, such as Google Docs, Google Sheets and others; it was discovered that there is a Google application for building dynamic web forms, named Google Forms. Seeing its flexibility and simplicity to generate Questionnaires, it fulfilled the needs of PLR. However, there was no API in C# that supported this application.

Nonetheless, during our research, it was also discovered that there is Google’s own JavaScript-like coding language, Google Apps Script\(^6\), capable of interacting and modifying all Google apps, including Google Forms.

Many Google apps, one platform in the cloud

![Image of Google apps]

*Figure 19 - Google Apps Script Blurb*

While it was far less than ideal to utilize an API within Post Lecture Review to run a Google Apps Script which would then generate the Google Form (Project’s Questionnaire), it was relatively simple to have it up and running.

The biggest challenges were properly understanding how Google handles the structure of projects and how to make the Script internet-accessible. In short, it was required to have a Google Project registered for the PLR Application, where the Google Apps Script would be associated to and become able to be published online as a “web app”. Additionally, the prototype would have to run an Authentication process, via Google OAuth2.

After fulfilling the required challenges, it was straightforward to have the PLR Google Script running. First, the Google Apps Script was developed, with all necessary parameters that are needed to generate the correct Questionnaire. Then this Script could quickly be tested via a debugging method and be published online. Now the PLR System can select a Lecture with any number of Topics and Questions, and run the Script using the Lecture information. Additionally, the Script call itself returns the Google Form URL, so the Professor has both a direct link to the Google Form and the ability to verify if the Questionnaire properly meets his expectations.

All the relevant PLR and Google Script code to generate Google Forms can both be viewed at the end of this section (Figures 20 and 21). This code can be run by any Google REST API call, also supported by C#, given PLR properly provides both the necessary Google Authentication and correct information to access the Script.

---

\(^6\) Using Google Forms within Apps Script: [https://developers.google.com/apps-script/reference/forms/]
function generateFormFromLecture(formName, id, count, title, topicString, questionString) {
    // Set up Form
    var form = FormApp.create(formName);
    form.setTitle(buildTitle(count, title));

    // Add Topics and Questions
    Logger.log(topicString)
    var topicArray = breakParser(topicString);
    Logger.log('Topic Array: ' + topicArray);
    generateTopics(topicArray, form);

    // Generate Page Break (form)
    var questionArray = breakParser(questionString);
    Logger.log('Question Array: ' + questionArray);
    generateQuestions(questionArray, form);

    // Comments
    //generatePageBreak(form);
    generateQuestionItem(form.addParagraphTextItem(),'Any Additional Comments?',false);

    // Generate Spreadsheet
    var name = id + count + title + GSheet;
    var sheet = SpreadsheetApp.create(name);
    form.setDestinationFormAppDestinationType.SPREADSHEET, sheet.getId());

    // Move files
    moveFile('WNXYrnsyCJ7f1KqwJF0FAvpXdKgo2', form.getId());
    moveFile('WNXYrnsyCJ7f1KqwJF0FAvpXdKgo2', sheet.getId());

    Logger.log('Form URL: ' + form.getPublishedUrl());
    Logger.log('Sheet URL: ' + sheet.getUrl());
    return form.getPublishedUrl();
}

Figure 20 - Google Apps Script, Form Generation Method

public static aps TaskString) generateForm(aps Lecture) {
    // Verify given lecture exists
    if (lecture == null) {
        throw new IllegalArgumentException("Lecture invalid");
    }

    googleAppsScript.services.appScript.run({
        name: "Lecture Request",
        version: "1.0",
        acquire: "default",
        parameters: [lecture]
    });

    // Generate Google App Script script
    var script = googleAppsScript.run.acquirePatternPatternListPatternPattern;
    var params = {lecture: Lecture, acquirePatternPatternListPatternPattern};

    // Create Google Apps Script request object
    var request = new googleAppsScript.run.LectureRequest(lecture, acquirePatternPatternListPatternPattern);
    request.lecture = lecture;
    request.acquirePatternPatternListPatternPattern = acquirePatternPatternListPatternPattern;

    // Create an execution request object
    var executionRequest = new googleAppsScript.run.ExecutionRequest();
    executionRequest.lecture = request
    executionRequest.acquirePatternPatternListPatternPattern = acquirePatternPatternListPatternPattern;

    // Execute the script
    googleAppsScript.run.executeScript(request, acquirePatternPatternListPatternPattern);
}

Figure 21 - PLR Script Call, Form Generation Method
5.4. Questionnaire Data

Continuing from the Google Forms implementation, it was possible to create and associate a Google Sheets file to each Form, so all the Response data is stored in a single file. To implement it took very little additional code, besides handling permissions so my Google Script can read/write Google Sheets files. After some quick testing and code adjustments, everything was stored in a Google Drive folder to hold all generated Questionnaires (Forms) and Response Data (Sheets).

Google has a Google Sheets API that can be used within the PLR System, skipping the need for Google Scripts to pass data. Every table entry on the Response Sheets basically generates a column for each existing field, which includes the Student ID, 1 to 5 ratings of Topics and text answers of each Question.

The biggest complication of handling Questionnaire Data was properly associating each respective Topics and Questions to the correct TopicResponse and QuestionResponse Models, but before any testing could begin, we began observing issues with Google Authentication.

5.5. Google Issues

5.5.1. Limitations

As the Post Lecture Review began accessing more Google APIs, the project began to fail every REST API call for Form Generation, where the OAuth2 either failed to complete or the authentication process had issues with the redirect IP of the project.

The first case was the hardest one, since we could not determine why it was failing from its vague return message. Any other documentation or relevant forums had similar issues with no responses. In the second case, the system would manage to properly generate the OAuth2 request, and then have it accepted by Google, but in the end, Google would fail to return the authenticated response. The error would refer to the Google’s inability to reach the assigned project’s response URLs, since any of the addresses assigned to our System’s Google Private Key would never have a matching redirect IP from the currently running instance.

5.5.2. Re-Evaluation of Thinking

Unable to determine how to correct either scenarios of this issue, development for Post Lecture Review stalled. After a week-long search, the following Google Issue Tracker topic about failed Google Scripts API calls was found, which basically determined that it has been an issue around the Google Scripts API since 2015. Additionally, in the documentation for Google Scripts API, there’s a warning that specifies that the API does not work with service accounts, which was the only method of Authentication that could run the Google Script from the prototype.

After some discussion with the Project’s Advisor, it was determined all Google-related implementation would be dropped, so PLR would generate its own Questionnaires, handling all routing and response data on its own.

---

Google Issue Tracker, “Execution API – can’t use service account” Topic: https://issuetracker.google.com/issues/36763096
5.6. Self-Designed Forms

Changing both perspective and architecture of the prototype, all Google related API calls and methods were removed, and the Database was restructured to remove all unnecessary variables and deprecated Models.

The first step to generating PLR’s own Questionnaires was to determine how it was stored, which led to the creation of the Response Model. This Model almost mirrors the structure of the Lecture Model, also having nested Topic and Question Responses Models. The major difference between the two is how Response objects all refer to its related Model, where Lecture holds information about itself and Response holds a Student’s response to a Lecture.

Subsequently, a Controller to handle both Questionnaire access and response logic was created, utilizing the following path for each Form link:

http://<website>/Forms/Respond/<Lecture ID>

This Forms Controller will generate a Form View for a Lecture when requested, determine if the Student ID exists and belongs to the correct Course, and consequently generate the Responses for each Form Submission.

5.7. Questionnaire Data II

5.7.1. Statistics Generation

The first step to drawing statistics in the PLR system, was to determine what sort of information would be useful to the professor about his own Course. Keeping that in mind, PLR was able to generate statistics on the given ratings for an individual Lecture, as well as, keep track of students that do and do not respond to surveys. With those in mind, it was possible to draw the following statistics:

1. Line Chart of all Lecture Ratings – To determine how Lecture quality progresses over time
2. Range Chart of all Responded Surveys – To determine how accurate the ratings of each Lecture were and inform which ones require more answers.
3. Bar Chart for Students with Most Missing Surveys – To find students which are missing Survey responses.

After determining which statistics would prove most useful, we searched and found a tool that we could use to display our Statistics with ease and simplicity.

5.7.2. Presentation

To draw and represent all statistics generated by Post Lecture Review, our project utilized Canvas.js\(^9\), which draws all needed graphs with ease and has a clean look. The usage of this API is quite straightforward\(^10\), either by downloading or referring to the website copy of the .js file, then utilizing the correct JavaScript calls to display whichever graphs desired.

The only challenge of utilizing Canvas.js is properly creating the Model which is utilized by Canvas.js charts to hold each individual single point of chart data, called Datapoint. Hence a new StatisticsController was made and dedicated to generating statistics in the form of Datapoint lists.

\(^9\) Canvas.js – Beautiful HTML5 JavaScript Charts https://canvasjs.com/

\(^10\) Canvas.js Documentation – Tutorials & Samples on adding Charts into ASP.NET MVC Applications https://canvasjs.com/docs/charts/integration/asp-net-mvc-charts/
as well as handle the pages related to it. However, the documentation\textsuperscript{11} of Canvas.js provides many different examples on how utilize its charts and how to modify those graphs to our liking, with only some limitations on expanding datapoints.

### 5.8. Server Hosting

After finalizing the implementation of Post Lecture Review, it was time to utilize the college provided VM, with its own PLR URL address. There were some previous minor tests with implementing a ASP.NET project within Ubuntu, where we did not find any major issues when hosting an empty ASP.NET project.

This section will explain some concepts of how ASP.NET projects are usually handled within Linux machines, where a more detailed implementation process is available at the Appendix 9.1 – Server Configuration, which includes both the process of how to host an ASP.NET project within Linux, and the issues and solutions we have encountered throughout the project.

#### 5.8.1. The LAMA Stack

The implementation of our server will follow the so-called LAMA Stack, which is basically a solution stack of software, normally free and open source, to run ASP.NET Core 2 projects. In this case, the LAMA acronym refers to the following software:

- Linux, being the operating system
- Apache, the chosen Web-Server application to host our application
- MySQL, for the Database management system
- ASP.NET, which is the framework that we are utilizing (written in C#)

This solution stack is a modification of the original LAMP stack, where the P could refer to a variety of different programming languages, like PHP, Python, etc. The LAMA stack will be explained in more detail in the following paragraphs, where it describes the different Software and their corresponding versions utilized in Post Lecture Review.

Having L as the first part of the Stack and having an Ubuntu (18.04 stable) as our operating system on college-provided VM, ASP.NET (C#) will not be natively supported. Which is why we will include Mono\textsuperscript{12} which simply is a platform that contains a C# compiler, the runtime application that runs the generated code in a compatible language with Linux, the .NET libraries necessary to our project and in addition some other libraries specific to C++/C# Development. Since Mono is included with the ASP.NET SDK that was utilized, we will only refer to the version of said package.

Secondly, we have A, the Apache Web Server implementation that hosts the ASP.NET project, handles all requests and access calls to the Server, and determines which refer to our application and what responses need to be sent back. To properly set up Apache to automatically host PLR and redirect all calls, it was necessary to disable the default Apache page and create a configuration file that describes the Request scheme and which ports are necessary for the application. Additionally, we created PLR’s own SSL files, to provide a more secure connection to the application.

\textsuperscript{11} Canvas.js – Documentation, Quickstart: Your First Chart in under 5 minutes
https://canvasjs.com/docs/charts/basics-of-creating-html5-chart/

\textsuperscript{12} Mono – Documentation: About Mono https://www.mono-project.com/docs/about-mono/
The following letter refers to MySQL, in which we had to both install and provide access to the application to create and access the Database that would contain all of PLR’s data. The only additional requirement for the ASP.NET application MySQL interaction is the MySQL Connector/Net, which is included with the ASP.NET Dev Kit.

Lastly, we have ASP.NET, where we install the Linux .NET Source Development Kit 2.2. Despite the fact we are not developing the application within Linux, it provides all necessary packages for Mono, ASP.NET, MySQL Connectors and other additional tools.

One last thing, which is not described within the LAMA solution, is the addition of a software called Supervisor\textsuperscript{13}, that provides automatic application run/monitoring and additional useful utilities not available to Apache. It is possible to host the application solely with Apache, however Supervisor allows to isolate the different parts related to Hosting and provides separate logs for the host and the application. One could argue this additional software changes the name of the LAMA stack to LAMAS, but there were no references to that name.

All software utilized is publicly available for use, and in continuous development.

The entire installation process for the server is available at Appendix 9.1 – Server Configuration.

5.9. Canvas Integration
After working on the PLR system and having our system accessible via URL, it was time to provide PLR within Canvas. After researching LTI and Canvas Documentation, as well as implementation examples and guides, we outlined the following steps to fully implement Canvas in PLR:

• 1\textsuperscript{st} – Access Canvas to create the necessary launch points so PLR is accessed by it.
• 2\textsuperscript{nd} – Determine the useful information given by Canvas at GET Requests calls.
• 3\textsuperscript{rd} – Utilize OAuth to both determine the user and load the correct Canvas Course.
• 4\textsuperscript{th} – Implement Canvas’ REST API to access and modify data within the Canvas Course.
• 5\textsuperscript{th} – Provide notifications to Students with access of PLR’s Questionnaires

However, due to time constraints, it was not possible to fully implement Canvas to our System. It was only possible to implement up to the 3\textsuperscript{rd} step, where there was some initial testing of Canvas’ REST API. Since the Canvas implementation was not fully completed, it was not included within the final deliverable, but the entire process of implementing and using it is fully described within Appendix 9.2. An explanation of the entire process with more details, such as code and Canvas configuration, is available at Appendix 9.2 – Canvas App Setup.

5.9.1. Learning Tools Interoperability
To integrate PLR with Canvas, we must fulfill LTI requirements. Those requirements are the following: (1) enforce HTTPS, (2) create our application’s XML Definition, (3) configure a Canvas course to import our application and at last, (4) generate the required Authentication to allow the Canvas connection to be completed.

The first requirement was already handled in the previous section, where we created PLR’s Server SSL certification and enforced Apache to utilize it. Thus, there’s no additional configurations for this step.

\textsuperscript{13} Supervisor – Introduction \url{http://supervisord.org/introduction.html}
Requirement 2 and 3 are entirely about generating an XML file describing PLR and providing that generated file to Canvas recognize PLR. With an LTI centered website which provided a form\textsuperscript{14} to generate the XML file to be run in Canvas. All fields to be filled are self-explanatory, with only requiring us to select Course Navigation and Account Navigation and fill both of their LTI Launch URL. With the resulting XML code, it was necessary to insert the code to Canvas, so it recognizes PLR as App.

It was recommended that this was done with a test/beta account, to make sure it doesn’t incorrectly affect any pre-existing Courses, but the account we utilized was brand new and made with intent of being used for PLR development. So, after logging into Canvas, we opened a new Canvas course and accessed its settings to open the Apps tab. There, we clicked the “+ Add”, which opened the following window:

![Add App window](image)

\textit{Figure 22 - Canvas’ External App “Add App” window}

Within it, we selected the configuration type (Paste XML), provided the App name. The Key pair, “Consumer Key” and “Shared Secret”, are utilized by Canvas and our application to authenticate users, hence we provided simple values like “ONE” and “TWO” respectively. At last, we inserted the XML code, which provided Canvas with all necessary data to reach PLR.

Now after we submitted our application, there was the PLR App listed under the External Apps page. Also, after leaving the Admin page, there was a new tab for the PLR App within the Course modules, which can access and open the PLR Index page and display its contents within Canvas.

\textsuperscript{14} Edu Apps – XML Config Builder [https://www.edu-apps.org/build_xml.html](https://www.edu-apps.org/build_xml.html)
5.9.2. REST API and its usage

To preface this section, anything described here and onwards was not implemented, so there are no examples of REST API implementation within the submission copy of the PLR project code. However, it was possible to generate simple GET calls from PLR to Canvas which provided some Course information, since the majority of REST API calls can be made with only a URL string. For more complex queries, all possible queries and attributes are detailed on the Canvas’ REST API Documentation.

First things first, we generated the proper OAuth authentication within PLR, to gain access to a specific Canvas course information. This was done with already existing OAuth libraries, in this case, IMS Global’s LTI OAuth library\(^{15}\) for .NET. This library simplified the whole process, so we only needed a single line of code to determine if it was a valid connection. Basically, it only requires a Boolean variable that results from comparing a LTI Library generated signature with the Consumer Secret from 5.9.1 against the Canvas’ request signature, which determines if it was a valid request.

After the whole authentication process was completed, it became a matter of generating GET/POST requests using Canvas’ REST API\(^{16}\). Due to time constraints, it wasn’t possible to further expand Canvas integration.


6. Analysis

6.1. Prototype Results

All testing of Post Lecture Review was done within the Virtual Machine provided by WPI, where it contained the entire project solution, with the omission of Canvas related code. Additionally, testing was done with the focus of determining the success of development of PLR, in terms of providing all mentioned features in the introduction of Section 5 – Approach.

In short, Post Lecture Review is an incomplete project, where it successfully managed to create a system to both manage and generate Lectures, as well as create Questionnaires for students to complete and handle all Response data from the resulting queries. Save some minor bugs, PLR completely provides a working Prototype of the ideal project solution. However, it wasn’t possible to complete the second part described in Section 5, with only partial implementation of Canvas’ REST API features. Nonetheless, utilizing ASP.NET allowed the development of PLR to achieve interaction with Canvas, albeit incomplete.

Despite already knowing that both LTI framework and Canvas REST API integration are fully functional and is utilized nowadays within commercial products, the development of PLR demonstrated that those were relatively simple to implement, albeit not fully functional, as a solo student project. This means that similar projects to develop a Canvas Application, or similar learning systems that follow the LTI standards, are not unmanageable and can better succeed with the lessons learned during PLR’s development.

Additionally, this technology demonstrates the possibility of integrating WPI applications to interact directly with Canvas, to facilitate data interaction between WPI and Canvas. This project shows even more possibilities for additional Major Qualifying projects on Canvas Integration, which could provide even more utilities that Professors could use to improve class experience, even including a more complete version of PLR itself.

For such reasons, PLR was successful in its development, despite having limited results. Most of these problems could have been addressed with additional people working, speeding up the development as well as focusing on separate issues instead of having to tackle all of them with limited information.
6.2. ASP.NET Core as a Tool

Development of Post Lecture Review had many difficulties due to the usage of ASP.NET Core, due to the unfamiliarity to this Framework and C#. Given the ASP.NET framework structure, implementations, approaches have evolved rather quickly, there’s far too much conflicting information about its many iterations and versions. This only made determining what different solutions work far more complicated. This also includes the latest version which I selected to develop PLR in (ASP.NET MVC Core 2.2), having much outdated or missing information on its documentation at the Microsoft’s Official website.

Additionally, ASP.NET also implements various technologies, such as Razor Views (for pages) and Entity Framework (for system-database data handling), that also evolve completely separated from ASP.NET and each other. Hence it was only fair to say that development was extremely difficult for a single person. Some of the biggest challenges included dynamic view generation with complex models, code-first table relationships and modification of its data, server hosting on a non-native language system, database duplication and generation, and many other smaller challenges.

But, from the perspective of someone with experience of working with Java EE on a professional level, there were many similarities to both the MVC structure and routing logic, to facilitate development. Additionally, it led us to recognize the many different improvements that the latest version of ASP.NET has made over Java EE’s approach to creating Web Servers, in terms of both technology and implementation. Which is why, we do believe ASP.NET is quite a powerful framework to master development on, with always increasing room for growth and adaptation as Microsoft continuously develops it.

These findings only highlight the importance of picking a stable version of ASP.NET with at least a year since its release, for an easier time finding the correct information as well more online discussion on the topic. This project began development at ASP.NET Core 2.0, slowly shifting versions forward, due to a few major complications which were only addressed in the following versions.

In summary, ASP.NET is an amazing ever-evolving framework for the development of server applications across the web, however the fast evolution of ASP.NET also greatly increases the difficulty of development, since there’s very limited correct information at each new iteration. The search of correct information only becomes more complicated due to the massive amount of conflicting information with different setups, framework versions, technologies developed independently from ASP.NET.
6.3. Versatility of Canvas APIs

The Canvas REST API is rather accessible, given it follows REST API standards, hence compatible with almost any application (able to make and with a lot of information regarding REST API usage. Manually building the XML application details file would have been somewhat difficult, but LTI’s own XML Generator made it simple. The biggest reason why the Canvas API wasn’t fully implemented was due to time constraints.

Too much time was spent on properly utilizing the REST API properly, being writing our own POST requests to modify data and access other utilities within Canvas. On the other hand, it was quite straightforward to make GET requests to pull data on all students of the Canvas course, however there was some authentication problem with POST requests.

It was originally planned to utilize the Canvas API for the following:
- Pull all Student information for PLR’s own Student Database
- Modify Student grades based off Form Completion on PLR (which was optional choice for the professor’s Course)
- Directly message Course Students with PLR form links, after Lecture Dates were met.

All these features are available to the REST API, but development stagnated when attempting to fix issues. There was plenty of information on making said requests, but nothing on debugging and properly determining the source of issues. Hence development shifted to making the PLR System functional, while unable to properly utilize Canvas.

However, the Documentation of Canvas is rather robust, having a large variety of external Apps developed solely to work in conjunction to Canvas. This serves to display it works, but we were unable to make it so.
6.4. Google Limitations as a Tool for System Development

Throughout the development of Post Lecture Review, we have heavily utilized different Google services, libraries and products to facilitate the development. Additionally, for the large variety of Google Products, there are currently (2019) 218 different APIs that are available to use with the most popular system development languages (like Java, C#, C++, Python, ...). That combined with the simplicity of creating Google Projects to have credentials for development, allows any developer to utilize its products within their projects.

However, PLR needed Google Forms, to create dynamic Questionnaires for students to complete and collect all response Data. This product does not have its own API, which meant that utilizing it for PLR would be no simple matter, or not possible at all. After researching the capability of modifying Google products, such as Google Docs or Sheets, it was discovered that Google had its own Scripting Language, which allowed developers to expand Google products beyond their original set of features.

This system is called Google Apps Scripts, where for every Google product and service available, there’s a corresponding API that works alongside Google Apps Script. Because of the large number of different products existing within Google, it is necessary to load each utilized API and its dependent applications to be able to modify them beyond their default scope or to work alongside applications that didn’t connect by default. But there is no point in being able to work with Google Forms programmatically via Apps Scripts, since it doesn’t work within our project.

Deeper into recursive API hell, there’s a C# library available to work with Google Apps Scripts. This API allows us to gain access within a Google Project, made before writing a Google Apps Script, and run any existing scripts within it. Given we managed to pass through the necessary OAuth2 Google Authentication and have the necessary Script information, we were able to temporarily work with Google Forms to generate PLR’s Questionnaires.

When we began working on the logic to get Google Forms data from a Google Sheets document, we attempted to use the already existing C# API for Sheets, since it would require less reliance on Apps Scripts. At this point, PLR began having non-stop issues with OAuth2 Authentication, whereas no matter what was attempted, it wasn’t possible to reverse any code to a point where PLR still was able to generate the necessary credentials. This meant a huge roadblock on the development of PLR, where months of work were essentially rendered useless and required a large change in our approach to Questionnaires.

After going through this experience, we determined that utilizing Google products without useable C# APIs is a dangerous risk, as standard Google documentation and most related information will not fully apply. And this, for a one-man project, was massive loss of code and time. Additionally, having to work with Google Authentication, there might be special circumstances between the different applications we are attempting to implement, which most likely will fail the authentication process. There was a possible solution to this issue with Google Authentication, but at that point we had spent far too much time on Google without any foreseeable solution.

As the closing notes on Google usage, the possible solution to our problem was hosting Post Lecture Review entirely within Google Project, fully circumventing the need for OAuth2 authentication, but it held too many unknown variables on whether it would work alongside
Canvas. This is a possible point of improvement in the scenario of future developers that would like to utilize any, if not many, Google products with their project.
6.5. Possible Improvements on this Project

The first improvement for this Project would be more students for a project like this, since it would provide major benefits to speed up development of a web application. One major benefit would be code reviews, where different people would inspect the same code and speed debugging. Additionally, different team members would be able to focus on different aspects of development, such as an individual specialized in Databases, another on APIs and so on. Focusing on specific sections of the project would allow a better understanding on specific areas.

Another improvement was to begin prototyping much earlier in development, which also would have provided great benefits, such as determining issues early and resolving the Database problems we ran into. One particularly nasty issue was the Database conversion between Visual Studio’s LocalDB and the VM’s MySQL, that required a lot of reworking late into development. Additionally, early prototypes would facilitate the professor’s understanding of the current state of the project.

The very last improvement would be to find a specialist on ASP.NET or C# at the beginning of the project, since most problems originated from unfamiliarity with both things. It was extremely difficult to find information on more complex code implementation or fixing related bugs. A large portion of this project was spent on debugging complex Models and trying to find specialists on the field, with no luck on the second.

Arguably, all points are related to development time, that would’ve allowed for a more complete development of PLR. Spread thin as the solo developer, it was hard to focus and master any of the many different aspects of ASP.NET Web App development.
7. Conclusion

In conclusion, the development of Post Lecture Review was a moderate success, given the system was fully implemented despite missing the Canvas integration. The most interesting notes to take out from this MQP would be the aspects of the development of applications integrated with Canvas, as well as Web Applications development in general.

On the matter of Canvas Integration, it is important to note the actual simplicity of reaching the minimum requirements for the LTI implementation. With us being able to display Post Lecture Review within Canvas, implementing the Canvas REST API was more challenging. However, REST APIs have been around since the early 2000s and there are plenty of guides on how to implement and which libraries facilitate their usage. Additionally, one can create a Canvas test account at no cost, allowing anyone to be able to develop applications for Learning Management Systems, due to the fact we followed the LTI standard. For a solo developer, a lot was achieved despite not reaching all set goals.

Web App development in general is quite complex, given it has been continuously evolving over the years. However, since many websites are shifting to deploying Web Applications instead of actual executable applications, there’s been more and more innovations coming out, to both improve development, provide more features and facilitate the entry of new developers. As someone that had his first experience on developing a Web App in Java EE (Enterprise Edition) with a product that was commercialized, ASP.NET provides incredible changes that weren’t available to me in Java EE. For example, both Database and Server configurations were far more complex, being beyond my capabilities to interact with within the company I have worked in. In ASP.NET, all that is required to get an application on the web is to create configuration files within the correct Apache folders and have it running independently at all times. Additionally, running tests in ASP.NET is vastly faster than with Java EE, where the downtime between tests to make minor adjustments has been drastically reduced. With only one person, PLR was deployed and available to any user within WPI’s network, in the matter of a week.

If anything, this project showed that developing Web Applications is worthwhile to other projects. With the rise of more and more Web Applications, having a good understanding of the structure of a Web Application would serve the developers well in the professional world. Post Lecture Review can be utilized as a stepping stone for either Canvas or Web App development, where most of the initial complications have been straightened out.
8. Glossary

API – Application Program Interface, a set of tools that provides the means for another application to interact with the API’s application.

ASP.NET – A unified web development model utilizing .NET framework, where software can be developed and deployed on the web\(^\text{17}\).

Canvas\(^\text{18}\) – Refers to Instructure’s LMS, in which PLR was developed to work in conjunction as an App. Most references to Canvas will refer to Instructure’s LMS, different from Canvas.js.

Canvas.js\(^\text{19}\) – A responsive HTML5 Charting Library with a simple API. Can be implemented in any HTML page and allows real-time changes. Utilized to generate PLR’s Statistics.

Consumer Key – Pairing Key for Canvas’ Authentication. This key is a key to be kept secret by the Consumer, preferably unique to the Consumer. The matching key is Shared Secret. Usually called the Private Key within Public-Key Cryptography.

.cshtml – File Extension of Razor View web pages. Allow the usage of C#, operations and references, that is compiled prior the page is sent to the accessing user.

Public Key Cryptography – A cryptographic system that uses key-pairs to encode and decode data. Anyone is capable of encrypting data using someone’s Public Key, but only the person that has the Private Key will be able to decrypt and read the data.

Git – A distributed version-control system for tracking changes in source code development. Github was utilized to host the git project publicly.

Github – An Open-Source Software Development Platform that hosts Git projects. This was the website utilized by the team to host all project source code, both for development and submission.

Google App Scripts – Google’s solution to rapid application development that either work with Google Projects, G Suite applications or standalone. It’s written in an extended version of JavaScript, with access to all APIs available to G Suite Applications.

Google Drive – Google’s own file repository system over cloud. Hosts all files generated using G Suite applications but can also contain any file extensions within the user’s available cloud space. All project files during Google’s integration were available at the team’s Google Account.

Google Forms – Google’s solution to simple Questionnaires generation and hosting. Allows to automatically get all form responses into a Google Sheets file.

Google Sheets - Google’s approach to a spreadsheet application, very similar to Microsoft Office’s Excel.

HTTP – Hypertext Transfer Protocol.

\(^{17}\) Techopedia – ASP.NET https://www.techopedia.com/definition/3213/asp-net
\(^{18}\) Instructure – Canvas LMS https://www.canvaslms.com/
\(^{19}\) CanvasJS - https://canvasjs.com/
HTTPS – Hypertext Transfer Protocol Secure, which utilizes SSL online Certificates to securely transmit data.

IMS Global\(^{20}\) – A non-profit collaborative that seeks to advance EdTech interoperability, innovation and learning impact. IMS determined the LTI standard to easily and securely connect learning applications and tools with learning platforms.

jQuery UI – A collection of GUI widgets, animated visual effects and themes implemented with jQuery.

JavaScript – A high-level programming language that is dynamic and weakly typed. It is part of the three core technologies of World Wide Web.

LMS – Learning Management Systems, which are web applications to assist both learning and teaching online.

LocalDB – A SQL-based Database available for Visual Studio, it is easy to install and requires no management. It is noticeably lightweight, reducing the need of constantly maintaining a database alongside development.

LAMA – A solution stack of software, usually free and open source, to deploy ASP.NET Core applications in Ubuntu. Based off its original counterpart, LAMP\(^{21}\) solution stack.

LTI – Learning Tools Interoperability is a standard utilized by LMS applications to share a standardized interface to incorporate external applications.

Mono\(^{22}\) – Open Source development platform for .NET Framework applications, which contains both development tools and infrastructure to run .NET in non-native Operating Systems.

MQP – Major Qualifying Project.

MVC – The Model-View-Controller is an architectural pattern that splits the components of a web application into the following: The Model, object definition and database implementation, the View, HTML web pages, and the Controller, routing logic and utilization of data/objects within the project.

MySQL\(^{23}\) – An Open Source Relational Database Management System, currently owned by Oracle and heavily influenced by Structured Query Language (SQL). MySQL natively runs on all platforms, often utilized in web applications and online publishing.

.NET Framework – A development framework written by Microsoft that provides a controlled programming environment, in which all developed software can be utilized in any Windows-based system.

\(^{20}\) IMS Global Learning Consortium – Home https://www.imsglobal.org/
\(^{22}\) Mono Project – About Mono https://www.mono-project.com/docs/about-mono/
\(^{23}\) Oracle – MySQL https://www.canvaslms.com/
OAuth – An Open Source standard for access delegation of web applications and APIs. Commonly employed to allow cross-website authentication, where you can utilize a website’s account in another website without providing access to their passwords.

PLR – Post Lecture Review, this MQP’s name.

PrimeUI24 – An expansion to jQueryUI, PrimeUI is a collection of rich JavaScript widgets very similar to PrimeFaces. It generates beautiful HTML components, with easy hookups to variables in the application.

Razor Views – An ASP.NET programming syntax to create dynamic web pages with C# and VB.NET programming languages. The file extension that Razor Views utilizes is .chtml.

REST API25 – RESTful web application interfaces (API) that utilize HTTP requests to GET, PUT, POST and DELETE data. REST stands for REpresentational State Transfer.

Shared Secret – Pairing Key for Canvas’ Authentication. This key is a key that is public and specific to the Consumer. The matching key is Shared Secret. Usually called the Public Key within Public-Key Cryptography.

SSL – Secure Socket Layer, a Security Protocol for HTTPS data.

Microsoft Visual Studio – An Integrated Development Environment (IDE) developed by Microsoft. Natively supports the development of all Windows supported languages, as well starting projects for all types of ASP.NET development.

VM – Virtual Machine that simulates a computer through an ISO file of an Operating System, PLR’s Virtual Machine utilizes Ubuntu 18.04 as its Operating System.

WPI – Worcester Polytechnic Institute.

XML – The eXtensible Markup Language is utilized to produce documents in a format that is readable by humans and computers. Specifically utilized in this project to generate PLR’s information for Canvas utilize.

24 PrimeFaces – PrimeUI - https://www.primefaces.org/primeui/
25 TechTarget – Definition of REST API https://searchmicroservices.techtarget.com/definition/RESTful-API
9. Appendix

9.1. Server Configuration

Our Server Configuration followed the LAMA Solution Stack, with Supervisor installed as an addition for better server management tools. This section assumes the developer was already provided with a VM running with Ubuntu 18.04, skipping the first step of the LAMP solution stack (L – Linux).

If unable to receive a VM from WPI, with the same Ubuntu edition, there will be an example of how to acquire a web-accessible VM machine on Amazon. The only requirement for utilizing Amazon is to have an account* which allows you to access the EC2 Instance Management to create the VM.

* To create an Amazon account for EC2, it will be necessary to provide Credit Card information to create said account.

9.1.1. Utilized Software Versions

<table>
<thead>
<tr>
<th>LAMAS Solution Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux – Ubuntu 18.04</td>
</tr>
<tr>
<td>ASP.NET – DOTNET SDK 2.2</td>
</tr>
<tr>
<td>MySQL – ver. 8.0</td>
</tr>
<tr>
<td>Apache – ver. 2.4.38</td>
</tr>
<tr>
<td>Supervisor – ver. 3.3.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Editing Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nano – ver. 3.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remote Connection Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>PuTTY – ver. 0.70</td>
</tr>
<tr>
<td>FileZilla – ver. 3.40</td>
</tr>
</tbody>
</table>
9.1.2. ASP.NET Installation

First step of this process is to get the necessary .NET packages to both build and publish an ASP.NET Core Project. Given ASP.NET is always in constant change, there are many different approaches to set it up, some being inaccurate or outdated. This project followed a few different guides\(^\text{26}\text{27}\text{28}\) in combination, adapting each guide’s parts as needed. The combination allowed PLR online and functional, hence the process that was followed will be described on the following pages.

To do so, we must first Register for a Microsoft key and feed, using the following commands on the machine’s Terminal:

```bash
wget -q https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
```

Next, we check the repository, then get the https package to allow for configuration of HTTPS requests. Afterwards, we update all available products for installation and install the .NET SDK package.

```bash
sudo add-apt-repository universe
sudo apt-get install apt-transport-https
sudo apt-get update
sudo apt-get install dotnet-sdk-2.2
```

Afterwards, it is possible to verify if .NET SDK was properly installed by using the following command:

```
dotnet --info
```

9.1.3. MySQL Installation

The following step is to acquire and install MySQL in the VM, which will be utilized to host the applications Database. Always make sure the Ubuntu packages are up to date, then acquire both MySQL Server and Client packages. Additionally, we will install the MySQL Dev package to assist configuration.

```
sudo apt-get update
sudo apt-get install mysql-server mysql-client libmysqlclient-dev -y
```


Since PLR itself generates and populates the MySQL Database during the PLR Database Update-Database NPM console call, it will not be explained at this Appendix section. Information on how this was done within ASP.NET is available at Section 5.2.
9.1.4. Apache Server

9.1.4.1. Installation

Now, Apache is the next step of the LAMA Stack Solution implementation. Apache will be used to handle all Server backend functionality and handle all requests sent to the PLR Server. Part of Apache and sometimes mentioned within this Section, Kestrel is responsible to all data sent and received to the VM. The functionality of Kestrel is often referred to as reverse proxy, which unscrambles all received packages and properly redirect them within the VM.

```
sudo apt-get install apache2
```

Simply by running the installation line above, it will automatically provide and HTML page for anyone that attempts to connect to the VM using the standard Port 80.

![Apache2 Ubuntu Default Page](image)

However, with the default Apache installation, it doesn’t initially handle the Reverse Proxy. Hence it will be necessary to enable the following apache modules, with the last command restarting the Apache Instance on the VM to apply the newly activated modules.

```
sudo a2enmod rewrite
sudo a2enmod proxy
sudo a2enmod proxy_http
sudo a2enmod headers
sudo a2enmod ssl
sudo service apache2 restart
```

The described modules in the code section from above does the following:\footnote{Apache, HTTP Server Project – Apache > HTTP Server > Documentation > Version 2.4 \url{https://httpd.apache.org/docs/2.4/mod/}}:
- **rewrite** – Provides a rule-based rewriting engine to rewrite requested URLss on the fly
- **proxy** – Multi-protocol proxy/gateway server
- **proxy_http** – Rewrite HTML links to ensure they are addressable from Clients’ networks in a proxy context.
- **headers** – Customization of HTTP request and response headers.
- **SSL** – Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols.
9.1.4.2. Secure Socket Layer (SSL) Generation and Configuration

Now that we enabled the SSL module within Apache, it will be necessary to provide an SSL Certificate so it is able to provide HTTPS links.

The first step will be to generate a Self-Signed SSL Certificate for Apache in Ubuntu.

```bash
sudo openssl req -x509 -nodes -days 365 -newkey rsa:4096 -keyout /etc/ssl/private/localhost.key -out /etc/ssl/private/localhost.crt
```

The command above begins the process of creating a Signature for the SSL Certificate, so the developer must fulfill the following fields, where one could argue the most important line is the **Common Name**, which refers to the server’s IP address. All additional information will be bundled up to the SSL certificate when the Server is accessed, so it provides information of either who is currently developing said Server or the company responsible for the SSL Certificate.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country Name (2 letter code)</td>
<td>[AU]</td>
</tr>
<tr>
<td>State or Province Name (full name)</td>
<td>[Some-State]</td>
</tr>
<tr>
<td>Locality Name (eg, city)</td>
<td>[]</td>
</tr>
<tr>
<td>Organization Name (eg, company)</td>
<td>[Internet Widgets Pty Ltd]</td>
</tr>
<tr>
<td>Organizational Unit Name (eg, section)</td>
<td>[]</td>
</tr>
<tr>
<td>Common Name (e.g. server FQDN or YOUR name)</td>
<td>[]</td>
</tr>
<tr>
<td>Email Address</td>
<td>[]</td>
</tr>
</tbody>
</table>

9.1.4.3. Configuration

The last step of the Apache Installation process was to configure Apache to recognize and run our ASP.NET code. Right now, it only runs on the VM, only serving the page displayed on Figure 23 - Apache's Successful Installation Default Page. Hence, we first must add the server’s fully qualified domain name to the following file:

```bash
sudo nano /etc/apache2/apache2.conf
```

Now, at the beginning or end of the `apache2.conf` file, simply add the following line:

```bash
ServerName localhost
```

The next step will be to generate a configuration file for the ASP.NET project we want Apache to handle, by determining some routing logic and necessary port actions to be taken. The configuration file will be called `plr.conf`, which will be created and open for editing with the following commands:

```bash
cd /etc/apache2/sites-available/
sudo nano plr.conf
```

Within said file, add the following content to it:
Now, after writing the above code into the file, save and quit it.

Without going in too much detail about the code written to the plr.conf file, this file determines how Apache handle different requests sent to it, by utilizing both ports 80 and 443 to handle the request. It is important to note that all requests are proxied at the root to port 5000 of the server, at the local address 127.0.0.1. This port also is utilized by both ProxyPass and ProxyPassReverse to allow for bi-directional communication.

Now it is time to save and utilize this configuration site, instead of the default one at 000-default.conf. Disabling that .conf file will remove the Apache Default Page, to be replaced by the ASP.NET application.

```
sudo a2dissite 000-default.conf
sudo a2ensite plr.conf
sudo apachectl configtest
sudo service apache2 restart
```

In steps, the commands above will do the following:
- Disable default configuration file (000-default.conf)
- Enable the recently made configuration file (plr.conf)
- Determine if the new configuration file is valid (should return OK)
- Restart the Apache instance with the new configuration

Now, by running the ASP.NET application at the folder it was published with the following command, it should open the application when http://localhost/ is accessed. Additionally, since the SSL certificate was properly configured, it will modify http://localhost/ to https://localhost/.

```
sudo dotnet run
```
9.1.5. Supervisor Installation and Configuration

From the previous step, you must have noticed that the server will only be available when the command `sudo dotnet run` is currently running on a terminal instance. Which is why it is required to now automatically open and run our ASP.NET application whenever the VM machine is active. It is possible to achieve said result by simply changing how the Apache implementation was done, however there's a utility available for Linux machines called Supervisor.

To install Supervisor, run the following command:

```
sudo apt-get install supervisor
```

However, Supervisor requires the application within the VM to be published to be able to run. To publish the application and make it available to supervisor we must do the following:

```
cd ~/plr
sudo dotnet publish --configuration Release
sudo mkdir /var/plr/
sudo cp -R ~/plr/bin/Release/netcoreapp2.1/publish/* /var/plr/
```

The code above will change to the ASP.NET source code folder and publish a Release version of the application. The generated files will be available at the folder `~/plr/bin/Release/netcoreapp2.1/publish/*`, which will be moved to a more accessible folder at the system's `/var/plr`, so it can be accessed and run by Supervisor. By accessing the `/var/plr` folder, you will see all compiled files of the published ASP.NET project.

The following step will be to create a configuration file that Supervisor will run the publish target folder application and generate all logs of both instance and application. The configuration file will be created and opened with the following line:

```
sudo vim /etc/supervisor/conf.d/hellomvc.conf
```

And will be filled with the following:

```
[program:dotnettest]
command=/usr/bin/dotnet /var/plr/PostLectureReview.dll --urls "http://*:5000"
directory=/var/plr/
autostart=true
autorestart=true
stderr_logfile=/var/log/plr.err.log
stdout_logfile=/var/log/plr.out.log
environment=ASPNETCORE_ENVIRONMENT=Production
user=www-data
stopsignal=INT
```

---

MQP-MXC-566

pg. 46
At last, we restart Supervisor which will run our ASP.NET application and stay active on the background.

```
sudo service supervisor stop
sudo service supervisor start
```

Utilize any of the following commands to read the logs generated by Supervisor for additional information.

```
# Supervisor logs
sudo tail -f /var/log/supervisor/supervisord.log

# ASP.NET Application logs
sudo tail -f /var/log/plr.out.log

# ASP.NET Application Error logs
sudo tail -f /var/log/plr.err.log
```

9.2. Canvas App Setup

To allow Post Lecture Review Application work within Canvas, it is necessary to provide the means for Canvas access PLR, as well as validate the connection.

Given the possibility the VM utilized for the development of PLR not being active, there will be the approach utilized to deploy PLR within Canvas. Hence the following instructions written in mind that the VM was set up and configured using the previous section instructions (Section 9.1 – Server Configuration).

The VM Public Address (within WPI’s Network) was “postlecturereview.dyn.wpi.edu” and was provided by WPI. However, if a new server is being set up, it will be necessary additional work for a public address that wasn’t done during development.

9.2.1. XML Config Builder

Before beginning the Canvas configuration, we will first generate a file that will contain most of Post Lecture Review server information, which will be used by Canvas to access the server. Thankfully, there exists a website focused in EdTech, which provides a XML Config Builder that we will utilize to generate the config file.

Within this page\(^{30}\), we must fulfill all fields available (except for Icon URL) and select the two following extensions, “Course Navigation” and “Account Navigation”. Also complete the empty fields of those two sections and generate the XML config file. Save this code to be utilized later during the Canvas Configuration section.

---

\(^{30}\) Edu Apps – XML Config Builder [https://www.edu-apps.org/build_xml.html](https://www.edu-apps.org/build_xml.html)
9.2.2. Canvas Configuration

First and foremost, it will be needed to create a Teaching Canvas account, which will be utilized to create a Canvas Course to insert our PLR App. Said account can be created for free at the following link https://www.canvaslms.com/ at any of the FREE FOR TEACHER links displayed at the main page.

After having an account created, it is time to set up a New Course in which the PLR Application will be made available. Within the newly created Course, find the Settings tab and open Apps.

Given PLR is a project developed only for the MQP, it is needed to manually add PLR as an external application. When pressing "View App Configurations" (outlined in Error! Reference source not found. below), the whole Apps window will change, removing said button and displaying a list with all Apps installed at this course (Error! Reference source not found.). Press the “+Apps” button to begin App registration.
Now, we observe the External App Add screen. However, it doesn’t provide the correct fields to provide the previously generated XML configuration. So, we must open the drop-down box for “Configuration Type” and select the “Paste XML” option, which we will fill with our PLR Application information and XML code (Figure 27).

---

**Figure 26 - New Display of the Apps section**

**Figure 27 - Add App via XML code**
In the Configuration Type, select "Paste XML" and use the XML code from Section 9.2.1 that was asked to be kept. Name the App as Post Lecture Review and use “PLRConsumerKey” and “PLRSecretKey” respectively for Consumer Key and Shared Secret. Those keys are utilized for user validation but are not currently utilized.

Now with the Add App Form complete, submit and reload the current page. Post Lecture Review should be displayed alongside any other Apps utilized within the course.

Additionally, Post Lecture Review now has its own link at the Vertical Menu Bar, where we reached the Settings page. To verify the App integration was successful, click the new Post Lecture Review link, which now loads the Index page of Post Lecture Review Application running on the VM.

9.2.3. Canvas REST API

Despite the initial Canvas Integration, the connection between Canvas and PLR is only one way, where Canvas is just browsing PLR. Currently, Post Lecture Review is unable to access and modify Canvas information, which would be needed to populate PLR’s Courses with students and lectures. Hence, this integration must now be made on the Application Code itself, where we will utilize Canvas’ REST API to perform basic HTML methods to interact with Canvas data.

In short, this section will only discuss both what a REST API is and how it works, with examples utilized from Canvas’ REST API. This is due the fact that this MQP did not successfully implement the whole REST API, mostly due to time constraints and prioritizing the base PLR system.

9.2.3.1. REST API Definition and Usage

To preface the definition of REST API, it is important to first understand why they are widely popular and able to be implemented in all kinds of Language Environments. REST APIs are essentially an API that follows the HTTP standard, capable of being implemented on any application that can connect to the web.

REST stands for REpresentational State Transfer, which was initially utilized to describe the HTTP Object Model, for early HTTP standards of early computing days. All REST operations are done via HTTP methods, GET/POST/PUT/DELETE, which follows a common pattern that any web application can follow and use.

REST APIs are constrained by the six following rules:

1. Uniform Interface – Which determines that the interface is always simple and separated from the application’s architecture.
2. Stateless – In short, the HTML request itself contains all the necessary information to both understand and execute the desired operation.
3. Cacheable – Simply means that clients can cache responses to prevent the usage of stale or inappropriate data in response to further requests.
4. Client-Server – Uniform separation between clients and servers, where both can be replaced and/or developed separately, as long the interface isn’t altered.
5. Layered System – Client does not have information on how the request is processed by the server, hence the server is able to implement its own methods load-balancing and shared caches.
6. Code on Demand (optional) – The ability of the server temporarily extends or customize the functionality of a client, given it does not break any of the previous constraints.

Clients utilizes the calls specified within its API, making calls using HTML requests through URI, query-string params, body information or request headers, or even a combination of those. For example, an URI and parameters would ideally be built like this `/users/12345` or even longer versions such as `/customers/33245/orders/8769/lineitems/1`, including the base URL with those URLs. Each call to the REST API will return a response code to determine the status of the call made, allowing the developer to expect different results and adjust his program to react accordingly.

9.2.3.2. Canvas’ REST API

To successfully use and implement the Canvas REST API within PLR, the best resources to determine what is needed to be done as well as you can do will be both the REST API Documentation and exploring the Canvas account created for the initial Canvas integration.

First, it is important to make sure all API requests to Canvas are done via HTTPS, against the normal Canvas domain. For this reason, PLR had to implement SSL to utilize HTTPS websites. Additionally, all Canvas REST API requests are following the JSON format, having HTML-JSON conversations displayed in the image below.

Lastly, all calls are done with OAuth2, preferably using HTTP Authorization header. API calls can also be made using Tokens sent to either the request Header or Query string.

Moving on to some examples, methods that would have been utilized for PLR will be described. The first example being utilizing Canvas Messages to pass all Students with the PLR Forms, after the corresponding Lecture was taught. For this example, there are many different ways of providing messages to students, including “Conversations”, “Announcements”, “Conferences” and “CommMessages”.

---

31 Canvas LMS – REST API and Extensions Documentation
If we were to utilize “Conversations”, it is possible to create a new Conversation with all students without allowed responses. The POST request would look something similar along the lines of the following:

```
https://canvas.instructure.com/api/v1/conversations?recipients[]=course_6426&subject="Post+Lecture+Review+Questionnaires"&group_conversation=false&body="plr_form_link"
```

Where each parameter is simply appended at the end of the base request, having the initial parameter to create a new conversation being `/api/v1/conversations` and all additional parameters added as needed.

Since the example above is a POST request, it would not return anything, but it is important to properly address the returned values within ASP.NET code. The GET request `/api/v1/courses/:course_id/modules/:id` would return the specified Module object as a JSON file, like the following:

```json
{
  // the unique identifier for the module
  "id": 123,
  // the state of the module: 'active', 'deleted'
  "workflow_state": "active",
  // the position of this module in the course (1-based)
  "position": 2,
  // the name of this module
  "name": "Imaginary Numbers and You",
  // (Optional) the date this module will unlock
  "unlock_at": "2012-12-31T06:00:00-06:00",
  // Whether module items must be unlocked in order
  "require_sequential_progress": true,
  // IDs of Modules that must be completed before this one is unlocked
  "prerequisite_module_ids": [121, 122],
  // The number of items in the module
  "items_count": 10,
  // The API URL to retrive this module's items
  "items_url": "https://canvas.example.com/api/v1/modules/123/items",
  // The contents of this module, as an array of Module Items. (Present only if
  // requested via include[]=items AND the module is not deemed too large by
  // Canvas.)
  "items": null,
  // The state of this Module for the calling user one of 'locked', 'unlocked',
  // 'started', 'completed' (Optional; present only if the caller is a student or
  // if the optional parameter 'student_id' is included)
  "state": "started",
  // the date the calling user completed the module (Optional; present only if the
  // caller is a student or if the optional parameter 'student_id' is included)
  "completed_at": null,
  // if the student's final grade for the course should be published to the SIS
  // upon completion of this module
  "publish_final_grade": null,
  // (Optional) Whether this module is published. This field is present only if
  // the caller has permission to view unpublished modules.
  "published": true
}
```
9.2.4. XML File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cartridge_basiclti_link xmlns="http://www.imsglobal.org/xsd/imslticc_v1p0"
xmlns:blti = "http://www.imsglobal.org/xsd/imsbasiclti_v1p0"
xmlns:lticm = "http://www.imsglobal.org/xsd/imslticm_v1p0"
xmlns:lticp = "http://www.imsglobal.org/xsd/imslticp_v1p0"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://www.imsglobal.org/xsd/imslticc_v1p0"
    http://www.imsglobal.org/xsd/lti/ltiv1p0/imslticc_v1p0.xsd
    http://www.imsglobal.org/xsd/imsbasiclti_v1p0
    http://www.imsglobal.org/xsd/lti/ltiv1p0/imsbasiclti_v1p0.xsd
    http://www.imsglobal.org/xsd/imslticm_v1p0
    http://www.imsglobal.org/xsd/lti/ltiv1p0/imslticm_v1p0.xsd
    http://www.imsglobal.org/xsd/imslticp_v1p0
    http://www.imsglobal.org/xsd/lti/ltiv1p0/imslticp_v1p0.xsd">
  <blti:title>
    PostLectureReview
  </blti:title>
  <blti:description>
    Post Lecture Review Application
  </blti:description>
  <blti:icon/>
  <blti:launch_url>
    postlecturereview.dyn.wpi.edu
  </blti:launch_url>
  <blti:extensions platform="canvas.instructure.com">
    <lticm:property name="tool_id">plr_mqp_lrlebrao</lticm:property>
    <lticm:property name="privacy_level">public</lticm:property>
    <lticm:options name="course_navigation">
      <lticm:property name="url">postlecturereview.dyn.wpi.edu</lticm:property>
      <lticm:property name="text">Post Lecture Review</lticm:property>
      <lticm:property name="visibility">public</lticm:property>
      <lticm:property name="default">enabled</lticm:property>
      <lticm:property name="enabled">true</lticm:property>
    </lticm:options>
    <lticm:options name="account_navigation">
      <lticm:property name="url">postlecturereview.dyn.wpi.edu</lticm:property>
      <lticm:property name="text">Post Lecture Review</lticm:property>
      <lticm:property name="enabled">true</lticm:property>
    </lticm:options>
  </blti:extensions>
<cartridge_bundle identifierref="BLTI001_Bundle"/>
<cartridge_icon identifierref="BLTI001_Icon"/>
</cartridge_basiclti_link>
```
10. Attachments

Latest Version of this project’s Source Code is available at Github.