Theses and Dissertations

2022-04-18

Playable Case Study Content Management System

Mitchell Stevenson Cross
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Engineering Commons

BYU ScholarsArchive Citation
Cross, Mitchell Stevenson, "Playable Case Study Content Management System" (2022). Theses and Dissertations. 9482.
https://scholarsarchive.byu.edu/etd/9482

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.
Playable Case Study Content Management System

Mitchell Stevenson Cross

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Justin S. Giboney, Chair
Derek L. Hansen
Amanda L. Hughes

School of Technology
Brigham Young University

Copyright © 2022 Mitchell Stevenson Cross
All Rights Reserved
ABSTRACT

Playable Case Study Content Management System

Mitchell Stevenson Cross
School of Technology, BYU
Master of Science

Educational simulations can help mitigate the natural gap between traditional education styles and the current professional world. Researchers at BYU have developed an educational simulation solution called Playable Case Studies (PCSs). PCSs are simulations that expose the user to real world scenarios and problems within a pre-built environment. With rising demand and use of these educational simulations, there is a need for easy and inexpensive ways to develop these simulations or Playable Case Studies (PCSs). We propose a content management system (CMS) that is tied to a system that utilizes dynamic modules that make up these simulations. We present a basic design and identify core functionality of the system. We include our results from utilizing this system and what future developments can enable the goal of easy and inexpensive development of PCSs.

In this study, we identify the design features needed for an easy to use, efficient content management system for educational playable case studies. We began by identifying both pain points in our old system and the features that would address these issues. We then designed and built a new system that included these features. In the testing of this new system, we primarily looked at the ease of building a new PCS with minimal technical knowledge. Although this project was a success overall, there are still points of failure due to the direct manipulation of JSON by content creators (who were not all developers), among other minor issues that need correcting. Overall, we found that our design was effective in providing an efficient platform for creating and maintaining PCSs.

Keywords: content management system, CMS, playable case study, PCS, alternate reality games, ARG, educational games
ACKNOWLEDGEMENTS

Throughout my graduate schooling I have received a lot of support and assistance. I would first like to acknowledge the funding that was received from a grant from the National Science Foundation that made this project possible (Grant No: 1915620). I would also like to thank the professors that helped spearhead this project: Justin S. Giboney, Derek L. Hansen, and Elizabeth Bonsignore. You provided the insight and help that allowed the development and more importantly the testing portion of this project to run smoothly and efficiently. I would like to thank all the other student developers that filtered in and out of this project. They provided support in developing and testing this project, allowing me to finish within the time frame that I did. Lastly, I would like to thank my family for the love and support that they have shown me throughout my schooling both in my undergraduate degree and especially my graduate degree.
# TABLE OF CONTENTS

LIST OF TABLES ......................................................................................................................... vi
LIST OF FIGURES ....................................................................................................................... vii

1 Introduction ............................................................................................................................. 1

2 Background .............................................................................................................................. 3

2.1 Educational Simulations/Games ...................................................................................... 3
2.2 Realistic Games and Alternate Reality Games ................................................................. 4
2.3 Playable Case Studies ....................................................................................................... 5
2.4 The Structure of a CMS ................................................................................................... 7

3 Methodology ............................................................................................................................ 8

3.1 Evaluation Methodology .................................................................................................. 9
3.2 Interview Selection ......................................................................................................... 10

4 Interviews .............................................................................................................................. 11

4.1 Interview 1 ...................................................................................................................... 12
4.2 Interview 2 ...................................................................................................................... 12
4.3 Interview 3 ...................................................................................................................... 13
4.4 Interview 4 ...................................................................................................................... 14
4.5 Interview 5 ...................................................................................................................... 14
4.6 Interview 6 ...................................................................................................................... 14

5 Playable Case Study CMS Requirements .............................................................................. 15

5.1 Analysis of the Previous Structure and Management of PCSs ...................................... 15
5.2 Findings .......................................................................................................................... 17
5.3 Needed Features ............................................................................................................. 18

6 Solution .................................................................................................................................. 20

6.1 Solution Objectives ........................................................................................................ 20
6.2 Solution Structure ........................................................................................................... 21

6.2.1 Laravel ...................................................................................................................... 22
6.2.2 Database ................................................................................................................... 24
6.2.3 Event-Driven Back-End ............................................................................................ 27
6.2.4 Front-End Dynamic Design ...................................................................................... 30
6.2.5 Data Management User Interface Design ................................................................. 32

7 Evaluation .............................................................................................................................. 35
7.1 Building the Risk Analysis PCS ................................................................. 35
  7.1.1 Risk Analysis Use Case ........................................................................ 36
7.2 Ease of Updating the PCS ........................................................................ 37
7.3 Results .......................................................................................................... 38
8 Discussion ........................................................................................................... 40
  8.1 Implications for Practice ........................................................................... 40
  8.2 Implications for Research .......................................................................... 41
  8.3 Limitations .................................................................................................. 41
9 Conclusion ......................................................................................................... 43
References .............................................................................................................. 45
Apendix A. Interview questions ........................................................................... 48
LIST OF TABLES

Table 2-1: Features Included in Current PCSs ................................................................. 6
Table 4-1: Interviewees .................................................................................................... 11
Table 5-1: Issues Needing to Be Solved .................................................................... 18
Table 5-2: Needed Features ......................................................................................... 19
LIST OF FIGURES

Figure 6-1: Laravel Router/Middleware Logic................................................................. 23
Figure 6-2: Entity Relationship Diagram (ERD) for Newly Designed Database............... 26
Figure 6-3: Process Event UML Flow Diagram............................................................... 28
Figure 6-4: Action Editor for a Chat Event ................................................................. 29
Figure 6-5: Initial Load of PCS UML........................................................................ 31
Figure 6-6: Data Management Interface - PCS Basic Information............................... 32
Figure 6-7: Data Management Interface - Events........................................................ 33
Figure 6-8: Data Management Interface - Event Dependencies.................................... 33
Figure 6-9: Data Management Interface - Module Content.......................................... 34
1 INTRODUCTION

Education today faces many challenges. It is primarily designed to teach the basic skills of a subject but often does not have any real-world connection and integration, resulting in a large gap between graduates’ understanding/knowledge and what is expected by employers [1]. Along with this gap, employers are noticing a lack of soft skills (“less tangible skills such as a positive attitude, dependability, team-oriented, well organized, effective communicator, and flexible” [2, p.1]) which are crucial to be successful during a career [3]. There are a few different socio-technical and technological solutions to these problems, among them is educational simulations.

Educational simulations address these challenges by providing unique opportunities for learning that are not available through traditional teaching methods. 1) They allow students to “experience” learning in a more immersive manner, which can lead to higher levels of student engagement [4]. 2) They provide a “safe place” to fail – particularly important in contexts that are high risk and rarely occur (e.g., a cybersecurity attack). 3) They are more likely to transfer to real-world contexts because they can be designed to be similar to real-world activities, tasks, and scenarios [5]. Despite some of the known benefits of educational simulations, they can be very difficult and costly to design and build. Thus, the need for a content management system that facilitates the creation of highly immersive educational simulations.

The goal of this project is to design a novel content management system that can support the creation of educational simulations to reduce both the time and resources required to build and
maintain these simulations. Structuring the frontend and backend to dynamically pull data from
the database allows the front-end pages and code to be reused by only making changes to the
data in the database.

The primary research objective of this thesis was to design and implement a content
management system that can support the creation of a variety of educational simulations
(specifically “playable case studies,” which are described in the Chapter 2). The associated
research questions include:

1) What are the design features needed for an easy to use, efficient content management
   system for educational playable case studies?
2) What architecture can support such features?
2 BACKGROUND

Recent surveys have shown that more and more students are preferring the online education model [15]. A large factor in the growth is the desire for remote learning [27,28,29,34]. With this emergence of online education combined with the need for remote solutions, requests for in-depth educational simulations have increased [16, 17]. Two main solutions for this newfound need are the creation of educational games and Playable Case Studies (PCS).

2.1 Educational Simulations/Games

Educational simulations can provide opportunities for authentic professional practices that would otherwise be difficult to experience. Gredler [6] establishes one type of educational simulation, “experiential simulations” that create a realistic environment, placing the player in a specific role within the given environment to complete varying responsibilities. Scholars and game designers have increasingly pushed to create more “authentic” practice-based simulations in recent decades. For example, Schaffër and colleagues [7] describe the benefits of what they call epistemic games, which link together ideas from games, educational simulations, and professional practice.

On top of creating a realistic environment for learning, educational games/simulations have proven to increase the motivation of the students to learn the material [18]. The immersion of
educational games creates a more realistic situation resulting in more motivation to fulfill the responsibilities the students have within the scenario. The key to epistemic games is conveying realistic scenarios and issues through the game. This immersion is essential to keeping the students locked into the experience. In regard to designing our own system, keeping all aspects of the experience going through our interface will help with this. Limiting the need for the subject to travel outside of the environment and focusing on having students accomplish authentic tasks using everyday technologies professionals use, we prevent the sense of This Is Not A Game from breaking.

2.2 Realistic Games and Alternate Reality Games

Meanwhile, game designers created unfolding, collaborative narratives that incorporate real-world technologies and practices in the form of Alternate Reality Games (ARGs). ARGs use “the substance of everyday life and weave it into narratives that layer additional meaning, depth, and interaction upon the real world” [8 p.6] creating a TINAG experience. TINAG is a key point in the effectiveness of education ARGs, creating an engaging experience and reinforcing the educational topics and material [5, 19], promoting long term recollection [9, 31]. Just like educational games/simulations, keeping this immersion is crucial to the effectiveness of the experience within ARGs.

Along with the immersion factor, ARGs also promote collaboration between participating parties [20,23,30,32]. Within an ARG, the user is tasked with the responsibility of shaping the story of their own play through along with their group members with each decision they make. Creating events and tasks that affect each other helps create this sense of control within the environment [24,33]. It is important to enable this control with the immersion into the
environment by eliminating the use of avatars. Having the users place themselves, and not an avatar, into the narrative of the game promotes the TINAG mindset [21,22,25,26].

2.3 Playable Case Studies

Similar to ARGs, Playable Case Studies (PCSs) allow students to gain experience and exposure to a specific professional field through the creation of an TINAG experience designed for an educational setting [25]. In a PCS, students are immersed in a narrative where they play a role in a company and complete a series of work tasks in a staged, but realistic, online environment [10]. PCSs are designed to combine both the immersion of ARGs, and the educational design found in typical professional practices embedded in epistemic games [4,7,11]. PCSs allow for exposure and realism of information for the desired field of work/study. They also let students gain experience with the real-world problems that are faced by professionals in the workplace [22,12]. This experience can then help students determine if that field is of interest and worth further investigation as a possible career choice.

To help build the desired narrative and to maintain the TINAG experience, each PCS needs certain features/modules to recreate the different resources used in the fulfillment of the emulated roles. Because we are trying to emulate real world communication platforms that are used in professional settings, the module requirement of each PCS overlaps. The previously constructed PCS narratives are proof that this overlap. Through developing these previous PCSs and interviewing developers that worked on those projects, we have found that there are basic modules, Table 2-1, that all the designed PCSs have: email, chat, documentation, and a pre/post survey [2,4,10,11]. Going through the different PCSs listed in Table 2-1, there is a familiarity
between the accessibility and functionality of the overlapping modules because of the specific need they fulfill.

Table 2-1: Features Included in Current PCSs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Cybermatics</th>
<th>Microcore</th>
<th>Spanish Museum</th>
<th>Risk Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Chat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Notes</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Modal</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Documentation/Resources</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Group/Character information</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Survey</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Machine Terminal</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clickable Map</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Article Editor</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exhibit</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Showcase</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

From the PCSs that have been tested with participating university classes, the narratives have been successful in creating an immersive effect and helping in the realism of the content the PCS was designed around [2,11]. Through the testing and evaluation of the Cybermatics PCS, there was a direct correlation between the interest in the field and the use of the simulation [10,22,25]. This further proves that PCSs have been successful in the pursuit of exposure to different fields of work and have helped in the determination of students searching for their current and future pursuits professionally [22].
2.4 The Structure of a CMS

There are basic needs and design features that are needed for content management system to function efficiently. There are four main features that a basic CMS requires: content modeling, content aggregation, editorial workflow and usability, and publishing and output management [35]. Content modeling is the method of structuring the content that is being managed. In selecting the format, or formats, that data is stored in, you want to make sure that it is one that meets the needs of what the data is being used for. For website data management, a very common format of data storage and sending is JSON due to its simplicity and readability [36].

Content aggregation is designing a logical way of storing data and connecting all the data together. This is typically done through databases due to ability of easily accessing the data and tying data together. Editorial workflow and usability deals with the ability to create and edit the data that is being stored and managed through the CMS. The CMS needs to have the ability to create new entries in the data and edit existing ones as well. The final requirement is publishing and output management. This deals with showing the data that is being stored. With website CMSs, this is done through the website itself. The typical design is to have a CMS interface that allows for the manipulation of the data being shown on the public facing side of the website.

After ensuring that the CMS has those four requirements there are several other features/functionalities that are standard. Designing a CMS for a website/webserver, there are specific ones that are more common, specifically: multimedia management, content file management, and form building and management [35]. These are just commonly found features; each CMS is unique and will require more customized designs to meet the needs of the system that it is operating on.
3 METHODOLOGY

As mentioned above, the system is missing some needed attributes and functionality. Through the initial design of this project, there was a strong emphasis on answering the question of what features we need to ensure the content management system would be easy to use and achieve our goals. We were able to pull from our experience with creating and managing PCSs from the previously built PCSs. A lot of the features that are included are pulled from the interviews with past and current developers, writers, and teachers/managers along with standard practiced features that are commonly found in content management systems.

Through my research, the best method I found for testing the solution to add in these attributes and functionalities is with the design science model. Design science is the process of designing a system in the context of the problem it is meant to solve [13 p.3]. This method consists of six steps: “1) identify and motivate the problem, 2) define the objectives of a solution, 3) design and develop an IT artifact, 4) demonstrate the use of the artifact, 5) evaluate the potential value of the artifact, and 6) communicate the design and significance of the artifact and findings” [10]. Step 1 was done in the literature review; Step 2 was done through the interviews and analysis of the previous system; Step 3 is described in chapter 6; Step 4 is explained in chapter 7; Step 5 is described in chapter 8; Step 6 is described in chapter 10.

As mentioned previously, there is a growing standardization of the use of educational simulations and games in class settings. In following the 6 steps of design science, the problem is
that this growing use is resulting in a need for easier ways to produce these experiences with little cost and effort. To solve this issue, an event-driven system with a non-technical, intuitive front-end content management system would achieve the goal of resolving these issues.

The design science method is the best fit to properly evaluate the efficacy of this proposed solution. To find the most efficient way to structure this event-driven system with the partnering content management system is a cyclical approach which requires many iterations of the same system, each one improving on the last. With each iteration, testing each artifact rigorously with current developers and content writers/managers to check the utility and efficacy [14].

Throughout the following sections, I will explain how the proposed solution meets each of the steps of Design Science Research.

3.1 Evaluation Methodology

In evaluating the success of the solution design there are two main methods. First method is building a new PCS from the ground up on the new system. Throughout the process of building this new PCS, detailed notes will be taken on the general process of developing every aspect of the PCS, the amount of help that is needed from developers, and how smoothly the PCS runs in play testing. This will be done through conducting play tests in college courses at both Brigham Young University and the University of Maryland.

Along with these play tests, I will be conducting interviews with developers and content managers familiar with this new system. These interviews will have the goal of evaluating the ease of use of the new system along with the division of labor and responsibilities between developers and data managers.
3.2 Interview Selection

Throughout the course of this project, there were two main points I wanted to focus on in the interviews. The first focus was the previous system and its pros and cons. For these interviews I wanted to get two perspectives: the developers and the content writers/managers. At the beginning of the PCS project those were the two jobs, but there was a lot of crossovers between the responsibilities of those roles. For these interviews I selected developers that were either involved in the initial design of the original system or developers that were highly involved in creating the first PCSs that were developed. I focused the questions on the process of developing a new PCS and the amount of time/process of helping the writers update and maintain the running PCSs, see Appendix Interview Questions for list of pre-written questions. For the writers, I selected writers that designed the narratives for the first PCSs due to them having to help manage the content of those PCSs. For these interviews I focused on the process of updating content for a PCSs and how it could be improved.

The second set of interviews was selected to be after the project was up and running. For these interviews I selected the same groups of workers, with the same types of questions. I modified the questions a little to focus on what can be done moving forward to further improve the system.
4 INTERVIEWS

The interviews that I conducted were very enlightening in both phases that they were conducted in. The interviews that I did during the research phase revealed a lot of features and functionality that the old system was missing. In the second phase of interviews, evaluation, I focused on ensuring the validity of the new system and ensuring that the issues needed to be solved have been resolved. As seen in Table 4.1, I was able to get an even spread of developers and data/narrative managers.

<table>
<thead>
<tr>
<th>Interviewee</th>
<th>Responsibility</th>
<th>System Familiarity</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Developer/Data Manager</td>
<td>Old</td>
<td>Zoom</td>
</tr>
<tr>
<td>2</td>
<td>Developer/Data Manager</td>
<td>Old</td>
<td>Zoom</td>
</tr>
<tr>
<td>3</td>
<td>Writer</td>
<td>Old</td>
<td>Zoom</td>
</tr>
<tr>
<td>4</td>
<td>Developer/Data Manager</td>
<td>Old &amp; New</td>
<td>Zoom</td>
</tr>
<tr>
<td>5</td>
<td>Developer/Data Manager</td>
<td>New</td>
<td>In-person</td>
</tr>
<tr>
<td>6</td>
<td>Developer</td>
<td>New</td>
<td>In-person</td>
</tr>
</tbody>
</table>
In conducting these interviews, each one lasted roughly 10-15 min and I started with pre-written questions that focused on the responsibilities and the process of building and maintaining PCSs (see Appendix). I would then ask to follow up questions to get a picture of what the exact responsibilities entailed. I wanted to get an accurate depiction of what a regular day looked like for them regarding developing new PCSs and maintaining old ones. I did my best to keep my questions open-ended as to not lead them to specific answers. All quotes from interviews have been slightly edited for clarity purposes. The following are summaries of each interview. There are quotes from these interviews in the following sections of the paper supporting the claims and findings found in the section.

4.1 Interview 1

My first interview was with a developer that was heavily involved in design and development of the first PCS. Through the course of the interview there were a few points that really stood out that were issues with their initial design in terms of long-term scalability. Those issues were that each PCS system was specifically designed and built for the needs of that PCS, there was no real way of testing the system before rolling it out for class use and updating the narrative information was a very technical process having to do it through a terminal.

4.2 Interview 2

The second interviewee is a developer that was brought on to the PCS project that worked up until just before we started developing this new system. In this interview, a lot of the previous points on the old system were brought up, along with the process of building new PCSs. It was made clear that this process was very inefficient. The process consisted of duplicating a template folder structure and manipulating the different modules to match the design that the PCS had.
We also talked about the process and methods for updating the narrative data. This developer’s answers supported the conclusion made from the first interviewee that this process is very technical. The process of bug fixing and updating narrative content on the old system was very tasking and time consuming due to the narrative data being split between the database and hard coded into the front-end pages. Another new issue that was brought up during this interview was the process of bug fixing. This process was very tedious because of the structure of making new PCSs. A developer would have to correct the issue in the template folder structure first, then go into each of the PCS folder structure individually and make the same update there. The final issue that came up was regarding user management. This developer noted that a lot of their time was spent handling user issues. Due to each PCS having its own login database and its own class system, it was a large issue having to manage all the PCSs on this front.

4.3 Interview 3

This interview was the last one done in the research phase. This was with a writer that helped with designing the narratives of the Spanish Museum and Cybermatics PCSs. The main point that was revealed through this interview was the process of making updates to the PCS narrative. Through the answers that the interviewee gave, it was clear that it was a very backwards process. The developer that oversaw the PCS would tell the writer what changes needed to be made to the PCS, the writer would then write the needed changes, and then the developer would implement the changes. This process needed to be changed to eliminate the need of the developer and allow the writer to do this entire process on their own.
4.4 Interview 4

The fourth interview was conducted in the evaluation phase and had two objectives. Because this developer was familiar with both the old and new systems, the first objective was to ensure that we identified all the issues with the old system. The second objective was to ensure that the design and implementation of our new system was achieving our goals outlined for this project. From the answers and comments given by this developer it was clear that our design was successful in meetings the goals of the project.

4.5 Interview 5

The fifth interview I conducted was during the evaluation phase. This interviewee is a developer that has helped build the new system along with acting as a liaison for some of the new PCSs being developed on this new system. From this interview it was clear that the system is successful in streamlining the process of developing new PCSs but does still require some fine tuning. Our initial design of the PCS manager interface was still too technical for our final goal. The interface was too reliant on the manager’s knowledge of JSON and its structure.

4.6 Interview 6

The final interview I conducted was with another developer that helped with building out features of the new system and has helped with the structuring of new PCSs on this system. Through this interview, it was noted that the system has made leaps towards our goals. Specifically, the dynamic design of the front-end modules. This developer also noted a weakness of our initial design of the manager’s interface, in its reliance on understanding JSON.
5 PLAYABLE CASE STUDY CMS REQUIREMENTS

Given that every content management system is for a unique system, each one has specific needs and requirements. For this specific CMS, I compiled a list of features needed to fulfill the functions for our system, see Table 5-2. I approached finding these functions through two different methods: analyzing the previous system and interviewing developer/data managers and writers that worked on the old system.

5.1 Analysis of the Previous Structure and Management of PCSs

The initial design of the PCSs that have been built; each had their own database specifically designed with the needs of that PCS in mind. Originally, this seemed like the correct approach to developing these educational tools. In managing both user data and data pertaining to the virtual experience, one simply needed to update the database pertaining to the given PCS. The core backend of current PCSs built with this approach use a Laravel structure connected to its own MySQL database. Each PCS web application is its own system, which makes things difficult for the development team, narrative managers, and users. Although having individual systems for each PCS provides the most flexibility, it has various downsides including developers needing to start an entirely new server with its own code and database, no central database for managing the narrative of the PCSs, analytics across PCSs is challenging, users
having to have a separate login for each PCS, and each of the many systems has to be updated and maintained.

Having this segregated design makes the task of creating a new PCS daunting. Developers have to build a new web application each time, customizing each page to be specific to the new PCS. There is no overlap in the files or the databases. This makes the task of creating a new PCS take longer than it needs to be than if everything was housed under one database and one folder structure designed dynamically. Along with making a new PCS, it also creates an unnecessary complicated process for bug fixing existing components. One developer familiar with the old system noted that it was frustrating having to update every PCS system with a bug fix instead of updating a singular page.

In developing a new PCS, the pages needed to be customized with the narrative progression data. The way the old system handle narrative progression was a checklist approach. A user would complete all the needed actions, in any order, before being able to progress to the next day. This made for an artificial experience and worked contradictory to the intended TINAG experience.

Managing the data for both the users and the narrative data for each PCS is more complicated than it should be due to each web application using its own database. To update the narrative data and/or any user data for multiple PCSs, the data manager must manually update each database individually. This requires someone that is well-versed in working with the technical side of the web application and querying the MYSQL database. In most cases, this task falls on a developer, increasing both the time it takes for the change to be implemented and the time it takes the developer to complete the current task he/she is currently working on.
The current design creates a hassle for users as well. If a user wants to play multiple PCSs they have to create a new account with each application, since each PCS maintains its own database. This forces users to remember multiple login information which can then force the data managers to deal with an increased amount of user updates.

5.2 Findings

In interviews conducted with developers and managers that worked on the previous system, it was very evident that needing a technical background and technical understanding to update the narrative data was frustrating. To make updates and changes to the narrative data, “You’d have to know how to access a database, log in, know how to navigate through the different, um yeah, navigate through the different databases and find the correct cell to modify it” (Interview 2). It also created a backwards process in updating the PCSs. In my third interview, the writer stated that instead of the normal flow of writers creating the content then it being updated on the PCS, it was the developers directing the writers on what they need written.

Through my analysis of the old system and the interviews, there were a number of issues that were obvious areas that needed improvement. Through my research, I have come up with ten main issues that need to be corrected with the new system, see Table 5-1. Through the course of the research phase of this project, I did note more than the ten included in Table 5-1, however, these ten issues were ones needed to create a baseline for the newly designed system. I used this list of issues as a checklist when I was designing and evaluating my new system.
Table 5-1: Issues Needing to Be Solved

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Managers interface to update PCS narratives is too technical</td>
</tr>
<tr>
<td>2</td>
<td>Creating a new PCS consumes too many resources</td>
</tr>
<tr>
<td>3</td>
<td>Require users to keep track of a multiple login</td>
</tr>
<tr>
<td>4</td>
<td>Duplicate front-end pages for PCS modules for each PCS</td>
</tr>
<tr>
<td>5</td>
<td>PCS progression is artificial, emulating a checklist</td>
</tr>
<tr>
<td>6</td>
<td>Each PCS needs its own database/server</td>
</tr>
<tr>
<td>7</td>
<td>No role distinction for users (student, teacher, admin, manager, etc.)</td>
</tr>
<tr>
<td>8</td>
<td>User progression is tracked differently per PCS</td>
</tr>
<tr>
<td>9</td>
<td>Class implementation varies</td>
</tr>
<tr>
<td>10</td>
<td>No live updating between group members</td>
</tr>
</tbody>
</table>

5.3 Needed Features

In order to make this improved, single system to create and manage PCSs there are a few things that need to be included. There are a lot of features that need to be included to solve major pain points in the old system, see Table 5-1. The main features needed in the new system are show in Table 5-2. Each one of these features will allow a key functionality that this all-in-one system needs.

The database needs to be redesigned so that it can track what PCSs each user has access to, what class(es) they are a part of, and the data pertaining to each individual PCS (both user and PCS specific). The front-end pages will need to be designed so that they utilize this data being pulled from the database and populate the page with the data pertaining to first the PCS, then the data pertaining to the logged in user. Lastly, the PCSs need to be driven by events that trigger
specific actions both in the front-end and back-end to progress the storyline to make the dynamic front-end pages possible.

Table 5-2: Needed Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Issue(s) Solving</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Single Server/Database</td>
<td>3, 6, 8, 9</td>
</tr>
<tr>
<td>2 Event-Driven Backend</td>
<td>2, 5, 8</td>
</tr>
<tr>
<td>3 Dynamic Front-end Pages</td>
<td>2, 4</td>
</tr>
<tr>
<td>4 Content Management Interface</td>
<td>1, 2</td>
</tr>
<tr>
<td>5 Multimedia handling</td>
<td>1</td>
</tr>
<tr>
<td>6 User Management</td>
<td>3</td>
</tr>
<tr>
<td>7 User Roles</td>
<td>7, 9</td>
</tr>
<tr>
<td>8 Page/Route Security</td>
<td>7</td>
</tr>
<tr>
<td>9 Websocket Implementation</td>
<td>10</td>
</tr>
</tbody>
</table>
6 SOLUTION

In creating this system, first my team and I created a backend system that is both event-driven and module-based to address issues 3 and 4 in Table 5-1. Second, we created a content management system (CMS) that is accessible from the front-end and simple enough that a non-technical teacher or content writer can utilize it in updating the narrative data, with a separate section for admins to manage user data and admin level PCS data. This was done to fix issue 1 in Table 5-1. We made sure to include, and continually add in all the features that would help make this system user friendly.

6.1 Solution Objectives

In continuing on to step 2 of the design science methodology, there are three main features that need to be included in the design of this new CMS. First, to resolve a majority of the issues we found, see Table 5-1, the database needs to be redesigned in a way to handle two main concerns: to handle and utilize a module-based system and be able to dynamically handle multiple PCSs that utilize the same front-end pages in order to reduce code and page duplication. Having a module-based front-end will allow developers to use pieces in multiple PCSs so that they do not have to manage multiple versions of the same page(s). The database should allow the system to dynamically handle multiple PCSs utilizing the same front-end pages by loading PCS specific data to populate these front-end pages.
Second, the backend needs to be redesigned to handle dynamic based narrative progression. The system needs a way to trigger updates to the front-end in response to actions the user performs as they work through the storyline. This will allow for a standardization of the PCS progression allowing the simplification of the narrative structure, making a standard management interface possible.

Third, the new design must provide a single sign-on, so users can access multiple PCSs without having to navigate to different web applications correcting issue 3 in Table 5-1. This will limit the number of login credentials the user will need to have access to multiple PCSs and allow for gaining/purchasing access to other PCSs in one location.

Fourth, the User Interface (UI) should be intuitive and simple enough for non-technical data managers to use to maintain the data for the PCS narratives, issue 1 in Table 5-1. This will allow the different PCSs to be updated without the need of a developer's help. The biggest interface that will allow for this is the event manager interface. Each PCS will be driven by a series of events and updating/managing the content and criteria for these events is crucial.

### 6.2 Solution Structure

To properly accommodate a CMS, the solution design needed to ensure that the system can be run from a single database. To achieve this, we needed a dynamic module-based system. The front-end pages load data in from the database that pertains to the selected PCS and display the PCS narrative specific data. This would in turn allow PCS managers to manage the data for all PCSs in one place.

To achieve this goal, a few things needed to be done. First, a foundation structure was selected for the rest of the system to be built on. Second, the database was redesigned to handle
dynamic data loading and storage. Third, the back-end was rebuilt from the ground up to be event-driven so that it could dynamically handle the different narratives for all current and future PCSs. Fourth, the front-end pages were (re)designed to pull the appropriate data from the newly designed database using the selected PCS as a key. Lastly, a data management interface was designed and built for the managers to maintain the PCS narratives, the teachers to manage their class data, and the admins to update the admin level PCS data and user data.

6.2.1 Laravel

To build this new dynamic system, we wanted to find a good foundation to work from instead of building the entire system from scratch. Laravel was the best option because it was designed for single page applications and it the fact it was used to build previous PCSs, making porting the previously built PCSs over to this new system in the future easier. Single page applications only have to load one main page and then they rewrite the data on that page when the user traverses through the application. This makes the load times for the front-end faster resulting in less load time experienced by the user. Laravel also provides easy integration with a prebuilt framework of four different sections: VueJs and blade for the front-end, the customizability and security that the Laravel router and middleware provides, the combination of a controller and state backend, and the easy integration of Npm and pre-built Npm libraries.

Because the front-end pages utilize VueJs, the front-end pages can dynamically load data from the database allowing customization of what is displayed to the user without needing to change the structure of the Vue page. This provides a clean, efficient way of creating reusable modules and solving issues 2 and 4 found in Table 5-1. To mitigate the strain on the developers when creating a new PCS, the front-end pages can be designed and built to be dependent on both
a JSON object that contains the necessary data for each PCS and computed variables. Computed variables can be tied to the state or to functions that are automatically updated and in turn update the front-end code. This allows PCS managers to create new PCSs without the need for developers, in turn making the entire creation process both easier and shorter.

Part of the Laravel framework is a pre-built router and middleware framework. There are two main routers that are used, the router containing the directions for accessing the different Vue pages and the router that contains the function calls for the different PHP controllers that the front-end uses to interact with the database.

![Laravel Router/Middleware Logic](image)

**Figure 6-1: Laravel Router/Middleware Logic**

As shown in Figure 6-1, any request to either router must go through the middleware framework. The middleware framework acts as the security gate to the backend of the application. When the front-end sends a request for either a Vue page or a controller method, the middleware checks the permissions for the user found in the database. This resolves issue 7 in Table 5-1, not having user roles and permissions, by distinguishing between the different roles and managing the access of each role. The middleware is a pre-built secure way to distinguish the permissions between the different roles that are included both now and in the future. If the
user has permissions to access the route requested, the backend pushes the request through. If the user doesn’t have the appropriate permissions, the backend returns an error code and the front-end handles the error code accordingly.

Another key built-in feature that made Laravel the best option for this project is the store/state functionality. This functionality provides a JSON object that is built by the backend and stored on the front-end providing a key functionality in addressing the issue of having duplicate front-end pages, see issue 4 in Table 5-1. Because Laravel is designed as a single page application, the front-end will load in all the necessary data pertaining to the user’s progress in the given PCS. When a user logs in and accesses a PCS, the backend compiles the data for each module included in the PCS and sends it to the front-end. This data is then used to populate the different Vue pages that the user visits throughout the storyline of the PCS. This will be crucial in tracking a user’s data for each individual PCS, allowing all the data to be loaded at once and easily referenced in the different module pages.

The last attribute is something that is not specific to this framework, but is something that the framework is built around, Npm integration. Npm is an open-source, software registry. It allows us to utilize libraries and VueJs components that are built by other developers to help ensure the dynamic build of the front-end pages. It also makes the process of building any new PCS modules and admin modules cleaner by utilizing well documented libraries that have been published through Npm.

6.2.2 Database

The database design was crucial to make the solution viable and address all our found issues, see Table 5-1. It needed to store the PCS specific content of each module while tying it to
each individual PCS. Because we used the Laravel web application framework to develop PCSs, there was already a foundation for the database. Having a single database that can distinguish between the users and PCSs was key in meeting a lot of the features for the CMS.

The users table original design included all necessary elements for the built-in login, but we also needed it to track the privileges each user would have in regard to the PCSs, issues 3 and 7 in Table 5-1. Thus, we added Boolean variables to track the user’s if they had admin, developer, teacher, and/or data manager permissions. Along with user permissions, we needed a way to track if a student user was part of a class, belonged to any class groups, and the role that they played in that class (teacher, student, TA), issues 9 and 10 in Table 5-1.

The structure for tracking the different PCSs and the modules included in each one was accomplished through multiple tables, issues 4, 6, and 8 in Table 5-1. We created a base table for PCSs and a table to keep track of what PCSs each user had access to. This table tracked the user’s role, base state, the access code used to gain access to a specific PCS, and if they were in a group with other students. To make the database dynamic, we included modules that were tied to each PCS. The base module table kept track of the module name and CSS file name. An intermediary table between this base table and the PCS table stored the PCS specific data. We also stored any user specific data for each module in each PCS. This table connected the module table with the table tracking what PCSs each user had access to.

To make the PCSs running on this system autonomous, we provided a way to store events that would happen both in the narrative and actions the user would need to do/accomplish, issue 5 in Table 5-1. These events would trigger subsequent events, progressing the storyline of the narrative. The database tracks these events, the dependency event(s) that needs to be completed before an event can be triggered, and the action that would take place on trigger of each event.
Figure 6-2: Entity Relationship Diagram (ERD) for Newly Designed Database
To analyze how intuitive the narratives of the different PCS designs were and for classroom grading purposes, we needed to have a way to see the progress of each user, issue 8 and 9 in Table 5-1. As a user plays through a PCS, they completed events and we logged the timing of these completions. We created a table to store this data, specifically the user’s id, the PCS id, the event id, and the time of completion.

Throughout the development of this system, small changes were made to our initial database design. As seen in Figure 6-2, the current design of the database handles the needs of the solution, but this is not the final database design. It will continue to change as more PCSs are added to our system that might require updates to tables and columns.

6.2.3 Event-Driven Back-End

Redesigning the backend to be event-driven accomplished two goals. First, it made the narratives more immersive by having the front-end react to specific actions that drive the narrative of the PCS. Second, it is necessary to keep this reactivity of the front-end and still have the front-end pages remain dynamic. Because the front-end pages are populated through data from the database, there needs to be a way to update this data when the user progresses the narrative of the PCS, issue 5 in Table 5-1.

This event system utilizes two main features: first, the controller functionality in the backend and second, the websockets Npm library which helps solve issue 10 in Table 5-1. Laravel makes integrating and utilizing both of these features possible. Along with these two features, the database tracks the progression and dependencies of the events tied to each PCS. The design of this system ties in with the different modules sending each one its own command. This takes the
load of updating the front-end off of the back-end by having each module responsible for updating itself with the passed in data from the back-end.

![Figure 6-3: Process Event UML Flow Diagram](image)

When an event is triggered in the front-end, it processes the event in the `process_event` function that sends the event id of the completed event to the event controller. The event controller queries the dependencies table to make sure the next event in the sequence is cleared to be triggered. If the next event can be triggered, it loops through all the actions tied to that event, sending each action to the corresponding module in the front-end with the updated data (see Figure 6-3). Each module then updates itself with this updated data.
The action variable tied to each event can have a number of different individual actions that are run on the front-end, updating the users state that then updates the data shown on the
different pages. Figure 6-4 shows an event with an action that triggers a chat message that is sent to the user. There is a specific chat message for each role for the PCS this event is tied to. When triggered, the system loops through these four actions and sends each one to the users front-end where it only runs the action(s) in which the user’s role is the same as the role to receive variable in the action.

6.2.4 Front-End Dynamic Design

Each page needed to be (re)designed to allow for content to be dynamically loaded from the database, addressing issue 4 in Table 5-1. Using VueJS with Laravel allowed us to populate each page with specific content dependent on the user and the PCS without having to hardcode any values. The loading of each page is slightly different if it is the initial load of the PCS. In comparison to the initial load, as seen in Figure 6-5, the only difference is that any following pages after the initial load do not have to load the state. Once the state is loaded, it is stored in the front-end and does not have to recompile from the database.

Vue allows for the manipulation and creation of HTML elements after data has been loaded from the database. Vue was chosen over using basic DOM because it is much more flexible working with the Laravel backend. It gave more control over both the timing of functions and backend calls and made the process of dynamically changing the page possible when the state change is triggered in the backend. From a developer’s standpoint, the Vue components made updating and creating the needed reactive pages for the PCSs very simple.
Figure 6-5: Initial Load of PCS UML
6.2.5 Data Management User Interface Design

With the new database design and the content for each module pulled dynamically, the content for each module is easily editable. To allow the PCS managers to edit these database entries, we created a simple page that loads all the module content and lets them view the data, issue 1 in Table 5-1. There are three main sections that managers need to update to effectively manage a PCS: general information for the PCS, PCS event data, and module content for that PCS. The general information consists of basic information about the PCS and the color scheme to use. Basic information includes the name, description, image, and visibility in the saved state. The PCS manager would have access to a page, Figure 6-6, that would allow them to manipulate this basic information for the PCS they were managing.

![Figure 6-6: Data Management Interface - PCS Basic Information](image)
Because the narrative is event driven, the database stores what each event is, the dependencies of each event that must be completed before it can be triggered, and the tasks that are displayed to the students in-game to help them track their progress. From the UI the managers have complete control over the event data, see Figure 6-7, and the dependencies for
each event, see Figure 6-8, allowing them to create new entries and adjust any old entries that need updating.

To allow the front-end pages to be used for multiple PCSs, they need data to be passed in that contains the differences in the data that the module(s) need for each PCS. The way we found that best works with our design was to store the content as JSON in a database due to the flexibility and ease of reading/manipulating it on the front-end. The module content section of the manager’s page, Figure 6-9, is where the manager controls the content for each of these different modules loaded in the PCS. All the module content is loaded into a table where the content JSON can be edited and saved to the database.

![Figure 6-9: Data Management Interface - Module Content](image)
7 EVALUATION

As stated in chapter 3.2, the success of this solution design was evaluated in two ways. The first was to build a new PCS from scratch and make sure that it ran smoothly. The purpose of this solution was to make creating and managing PCSs simpler and easier. We tested this outcome by building a new case study (the Risk Analysis PCS) using our new solution. Second, we evaluated the ease of managing and updating the PCS’s narrative by the managers. This was to ensure that the solution made it easier for managers to update the storyline of the PCS and that they did not need help from a backend developer. This would eliminate the need for an experienced developer to be assigned to updating a PCS with the desired changes made by the managers.

7.1 Building the Risk Analysis PCS

In building the Risk Analysis PCS, which was the first PCS built on this new system, there were a lot of front-end pages that were built or redesigned to work with the new database and backend structure. Because previous PCSs had already been built, front-end pages from those could act as a base for the new pages that were developed. Because this was the first case study built on the new system, the development was not a reflection of the desired result from the solution because it required more adjustments on the front-end pages. In future PCS production, there will not be a need for major changes to the front-end pages and it will all be controlled by updating the database, which was the end result of this solution.
After the development of the front-end was completed, the PCS managers had full access to update and change the content of the PCS. They could make changes to any events triggered throughout the narrative from a manager’s page that showed the module content, events, dependencies, and basic information for the Risk Analysis PCS. This allowed for the division of work between the developers and the managers, letting the developers focus on building new modules for future PCSs and the managers focus on updating the PCS content and narrative to accurately create immersive experiences to facilitate the desired learning environment.

7.1.1 Risk Analysis Use Case

Once the PCS was at a stable state, there were a handful of classes from both BYU and Maryland that ran through it as a part of their curriculum. As they played through the Risk Analysis PCS, we tried to have a developer present taking notes on bugs that were found and helping with any technical issues that the user might be having. The classes successfully made it through the narrative of the PCS, working with a developer to get through some minor bugs that were found.

With this being the first PCS developed on the new system, there were a lot of bugs to be fixed. To do this, there was a lot of collaboration between the managers, developers, and the teachers that were willing to test the PCS in their classes. As bugs were fixed, the desired division of labor became more and more clear, allowing the developers to focus on future PCS development and allowing the narratives and content to be completely managed by the PCS managers or writers.
7.2 Ease of Updating the PCS

With the successful separation of labor between the developers and the managers, it was important that the interface was intuitive and easy for the managers to utilize. Once the front-end pages were completed and extensively tested, the managers had complete control over the PCS content. However, before this control was fully given, the interface needed to be tested. To do this, we had one of the heads of the project, familiar with the backend design, work with the narrative designers to make the desired changes.

In the user testing with the classes that agreed to test the system, a lot of content bugs were found mainly with the event processing portion of the system. To debug these issues, the manager’s interface was used to update the database to correct these minor syntactical errors in the JSON that controlled the content shown/triggered.

The initial design of the manager’s interface solved these issues, but it did require an understanding of the backend. The managers interface requires additional updates to eliminate technical errors that arise from the use of JSON to pass data to the individual modules. Once the test was over, I interviewed those responsible for updating the narratives on the interface that we designed. Through these interviews it was evident that “…the CMS makes it is easy to modify the narrative content” (Interview 4) and is a more user-friendly system for the PCS managers. There were still some areas that were a little technical and required an understanding of JSON structure. In creating a new PCS, one of the developers that have helped with the new system stated that “…building it from the ground up does take knowledge of JSON, you got to know JSON structure” (Interview 5). These areas are focused on and having new interfaces designed to eliminate the bare JSON structure.
7.3 Results

In the course of the user testing on the system, the solution did achieve the goals that it was designed to achieve. But with all new systems, there were points of failure that were found, mainly in the interface for the managers. Because the system runs primarily off of JSON, the managers need the ability to update these values within the database. However, the sensitivity of certain characters within the JSON made it difficult to ensure that it still functions as intended. To correct this, a new editor should be incorporated to help eliminate the syntactical errors that come with using JSON by making the process even simpler to create and edit these JSON variables stored in the database.

Another issue that was discovered during user testing, was problems with the user and class registration/management and the tie between the users and the PCS access they had. Some users had issues with registering for access to the Risk Analysis PCS for a certain class. This resulted in users not seeing the list of active PCSs, events not processing properly, and the inability to join a class/section.

Along with those two main issues, a few minor user interface bugs were found in the actual PCS itself. A few of the modules did not account for the variety of thought processes that a user can have. In certain modules, users were able to bypass crucial events which resulted in the halt of their progress within the narrative. This is something that was relayed to the developers and was corrected to eliminate this error for future users.

Along with the tests already mentioned, I performed interviews with developers and data managers on the changes the new system has. Through these interviews, it was clear that there is still a small blur between the responsibilities of the developers and writers in regard to creating a new PCS, primarily in regards to the events for the PCS. To populate the events for a new PCS,
the manager must understand how the events work and that “certain events need to trigger
themselves” (Interview 6). Because of this, there is a “barrier to get up to making a dependency
and events” (Interview 6). This is because the features designed for non-technical users are still
being refined. The functionality is there, but the interface is currently being refined and
simplified for those who do not have a technical background. All the other interfaces (admin,
manager for updating, and teacher) have been very successful in providing a clean interface to
achieve the necessary responsibilities of the respective role and have allowed developers to focus
more on bug fixing and developing new modules which is one of the main goals of this project.
8 DISCUSSION

While designing and testing this system, there were a number of things that came up that caused us to go back and redesign sections of the system. We learned a lot about the best way to structure the system, features that would be useful to add in the future, and some unforeseen limitations that we faced in this implementation.

8.1 Implications for Practice

In building a system like this, be sure to test for edge cases in the features being included. One of the big issues we realized near the end of the project was how we structured the PCS access. We have it designed to be where each user has a single access point to a single PCS. However, through our testing we found that that was an issue with students retaking a class that had a PCS as part of its curriculum or if they had multiple classes that used the same PCS. We concluded that the system should track a user’s progress not just by PCS but by class and PCS. This is something that is planned to be implemented but is a small flaw in the current build of the system we built.

In a system like this, it is crucial to plan out the full design of the system as much as you can before you implement it. We spent a month or two designing the database and the backend before we built a lot of it out. This made things easier later on when we started combining different sections that different developers were working on. Given a project is going to morph...
as you build it due to unforeseen problems arising, it was very helpful to have a foundation
designed and thought out before building it. In this design phase, it was helpful to have someone
play devil’s advocate. It made sure that a topic was viewed from as many angles as possible,
showing problems that could exist as unlikely as they are.

8.2 Implications for Research

Moving forward, in our future research we will treat each of the current and future
developed Playable Case Studies as individual case studies for the Content Management System
(CMS) that we have developed through this project. Along with continuing to improve the
system and its functionality, for each individual case study, we will be looking at the amount of
updating required on the front-end pages for each new PCS, the ability to have the PCS
managers build out and manage the narratives without help from the developers, and the
intuitiveness of our user interface for the managers through direct feedback from them.

Along with continual monitoring of the system, there are a number of features that could
be implemented to further the functionality of this system. A research role could be added that
has permissions to view the PCS data pertaining to users that had agreed to be included in
research data. Specific functionality could be added to the teacher/admin pages allowing them to
perform batch downloads of user data including emails sent on the PCS email interface,
documents submitted through the PCS, and more.

8.3 Limitations

This project had a number of limitations through the course of developing and testing it.
One of the biggest was the number of developers and managers that were working on and
utilizing the system. Because this project is developed as part of a research project at a university and is being funded by research grants, the number of developers we had on it was minimal and there was high turnover. Because the developers were students, we had a decent number of new developers brought on to replace graduating developers. Having to introduce a new developer to how the system was structured and how it worked would slow down the progress we were making.

In our testing, we were limited to a class or two in both universities (BYU and Maryland) working on it. It was helpful to stress test in a full class but having only a few classes run through the new PCS built on this system leaves many bugs undiscovered. In testing a new system, it is crucial to have as many people test it as possible. Each person thinks through things differently and performs tasks in a slightly different manner. These variances help discover bugs that the system may have, allowing the system to be even more polished.
9 CONCLUSION

With the growing area of educational simulations, the need for systems that can generate these experiences to enable this type of education is crucial. With this and basic design principles in mind, we set out to find what features are needed within a CMS to support the creation and maintenance of these Playable Case Study experiences. We created a user-friendly system that implements these features and allows PCSs to be created with minimal development, allowing components to be used in multiple PCSs without need for adjusting the code.

In this study, we set out to identify the design features needed for an easy to use, efficient content management system for educational playable case studies. Through our initial research, we identified 10 issues that the old PCS system had, see Table 5-1. This list was used as a checklist in our evaluation of our newly designed system. After identifying these pain points, we selected and designed features that would resolve these issues, see Table 5-2. We then designed and built a new system that addressed these points. Once this new system was at a stable, functioning state, we began testing it through developing a new PCS purely on this new system and ran play tests on the newly built PCS.

Through the testing we were primarily looking for the ease of building a new PCS with minimal technical knowledge. Through the course of our testing, we found that 1) there are still points of failure due to the direct manipulation of JSON by content creators (who were not all
developers), 2) having dynamic front-end modules does support our goal of making new PCSs
easier and less time consuming.
REFERENCES


APENDIX A. INTERVIEW QUESTIONS

For Developers
familiar with old system:
- How long did it typically take to create a new PCS?
- What was the typical process of creating a new PCS?
- How much code would be duplicated/pulled from a previous PCS?
- How many different servers/systems did you have to manage/update?
- What are the strengths and weaknesses of this system?

familiar with the current system:
- How long does it typically take to create a new PCS?
- What is the typical process of creating a new PCS?
- How long does it typically take to make a new module?
- What are the strengths and weaknesses of this system?

For Managers
familiar with old system:
- What was the process of updating a small task in a PCS narrative?
- Did updating a PCS’s narrative require development experience and familiarity with the back-end?
- How much paid time would it take to update PCS narrative information?
- What are the strengths and weaknesses of this system?

familiar with the current system:
- What is the process of updating a small task in a PSC narrative?
- Did updating a PCS’s narrative require development experience and familiarity with the back-end?
- How intuitive is the CMS front-end?
- What aspects did/do you struggle with and what can be improved?