Department of Mechanical and Mechatronics Engineering

Developing New Features for Versio Content Portal

A Report Prepared For:

The University of Waterloo

Prepared By:

Evan Hum

January 17, 2022

Table of Contents

List of Figures

List of Tables

Summary

Over the past co-op term, the student worked closely with team members on implementing new features for the product known as Versio Content Portal. The student's objective was to complete two cases related to Content Portal in order to prepare for the next public release. The cases that were worked on were VMGMT-12102: Add the CTRL + ← and CTRL + → shortcuts to the proxy player, and VMGMT-11963: Update CP automated tests to use domain users.

VMGMT-12102 required new shortcuts to be added to allow the user to move through segment and marker boundaries more easily. The student successfully implemented an initial version that allows the user to do so by creating a list of frame objects. Each frame object contained a frameNumber for comparing to the current frame, a markerIndex for selecting the marker, and an id for determining which marker the frame object belongs to. Using this list, the student could determine which boundary the user was currently on, and the next closest boundary by matching properties. However, it was missing some additional features such as using the shortcut while the preview was playing and moving from frames in between the boundaries.

VMGMT-11963 required domain user parameters to be added to the automated tests. This would allow for other teams to be able to test and use their accounts instead of having to integrate the hardcoded accounts. The student solved this case by adding two parameters, username, and password, for each of the four user types as well as a parameter for when the user only enters a single type. Authenticated test files were updated to check if their user type was inputted, and if it was, they would add their test suite to the automated tests. If their user type was not added, those tests would not run. Due to difficulties with determining the number of user types inputted, the automated tests only support either one user type or all four. The next steps for this case would involve implementing two and three user type inputs so that the user has the freedom to test as many or as little users as they would like. Furthermore, the test files need refactoring to reduce the amount of non-authenticated tests included in authenticated test suites and finally, the report file for reviewing the results should prevent tests that did not run from showing up in the report as pending.

Overall, the student completed the two assigned cases but discovered new improvements and features that could be added before the release. The release is not scheduled to be given out yet and so the updates are being passed on to other team members.

1.0 Introduction

This report will focus on the work the student has performed surrounding Imagine's Versio Content Portal service. Over the course of the term, the student has been assigned many cases involving implementing new features, fixing existing bugs as well as creating cases for newly discovered bugs. This report will go into detail regarding two of the main cases that were worked on for developing new features in Content Portal.

1.1 Description of Content Portal

Content Portal is a library that manages all the company's assets. This includes all their clips, audio, images, and any other content type they might have. These assets are used to plan out what is going to play on television. Companies can do so by building schedules in Content Portal and previewing all their clips before they go live on air. Content Portal also helps with organization through sorting, filtering and custom grouping features which can all be done using the Content Portal interface (see Figure 1) [1].

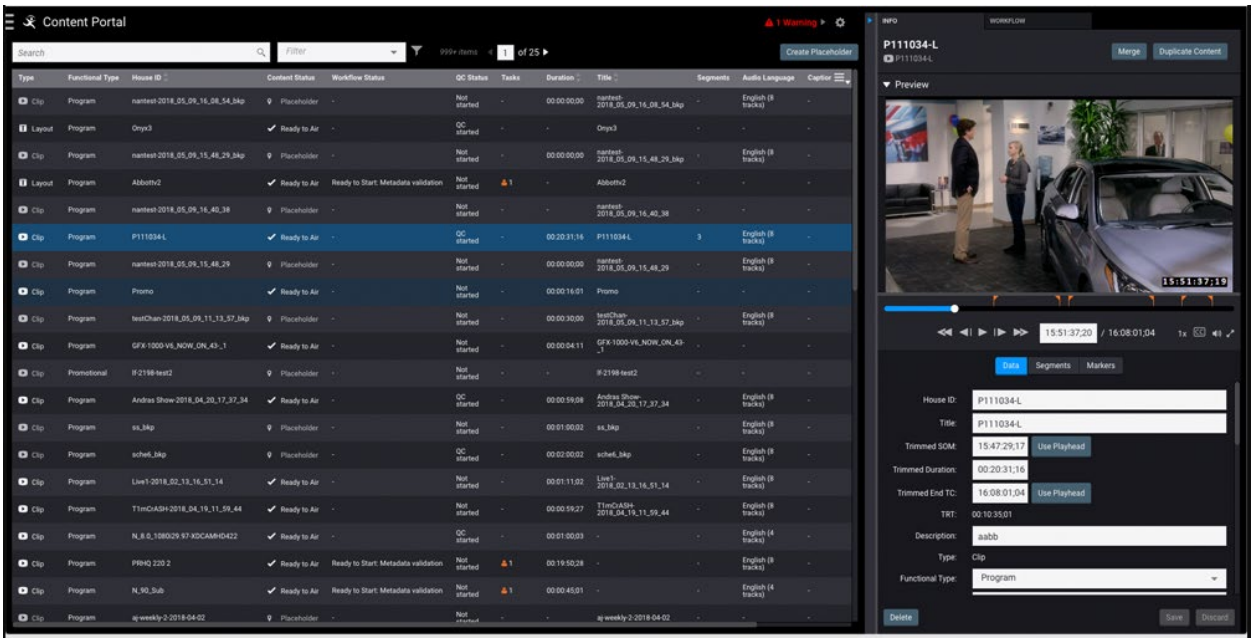


Figure 1: Content Portal Interface

The team that manages Content Portal is known as the Dropframers which belong to the Versio Management team or VMGMT for short. The team has biweekly iteration planning meetings where they decide which cases are the highest priority and which can be saved for later. Other considerations include the capabilities of the members and what areas they work in. The student found that they made the most progress with cases related to Content Portal and often looked for cases in that area.

1.3 Objective

The objective of this project is to complete the assigned cases so that the next version of Content Portal, known as the 4.7 GA, can be released on time. GA stands for General Availability and expresses that the specified version (4.7) is now openly released to the public [2]. The two main cases that were focused on and will be discussed were named VMGMT-12102: Add the CTRL + ← and CTRL + → shortcuts to the proxy player, and VMGMT-11963: Update CP automated tests to use domain users. The following numbers are the case number which will often be referred to as the name of the case. CP stands for Content Portal. As an example, the case description of VMGMT-12102 can be seen in Figure 2. More details will be provided for these cases in the following section.

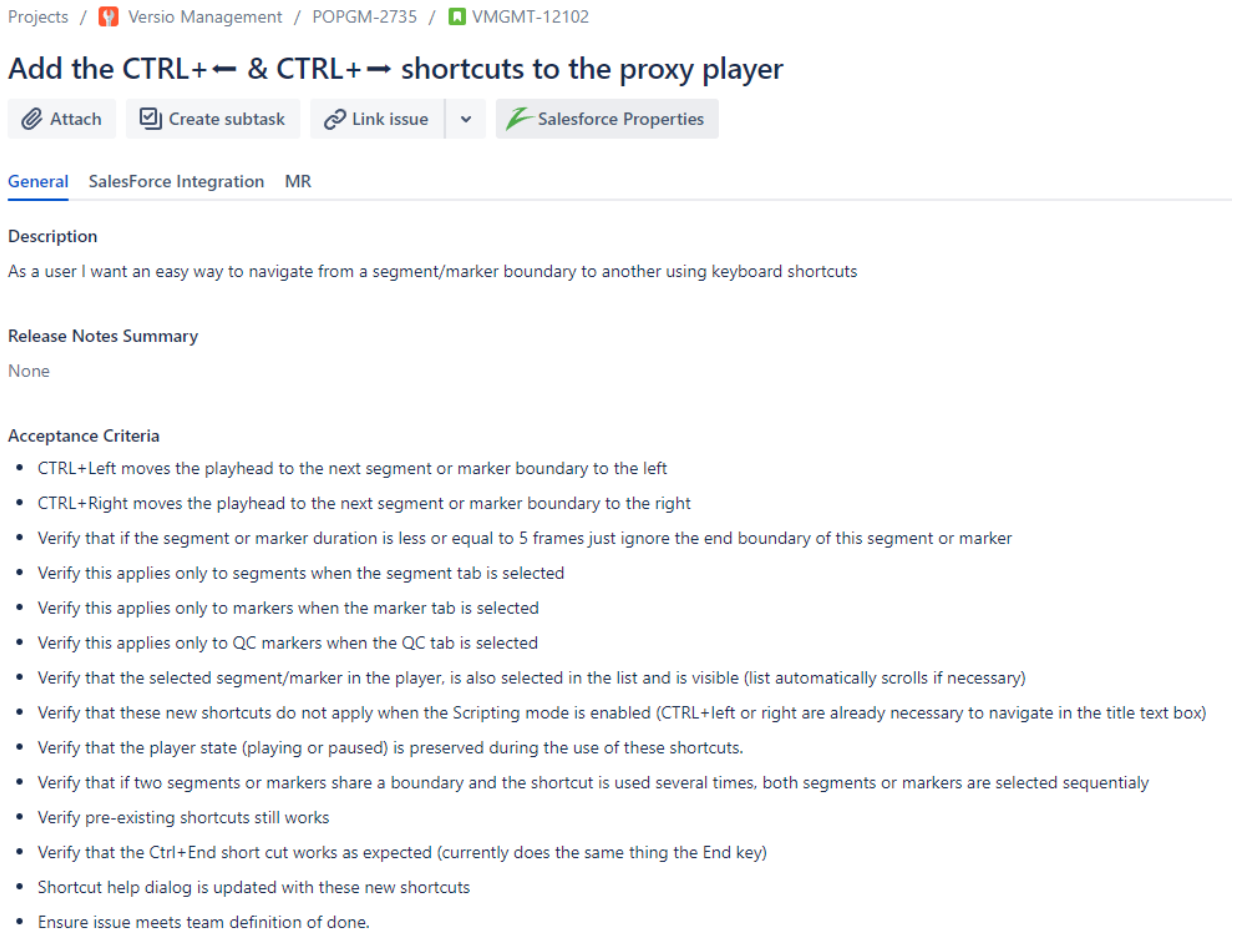


Figure 2: Case description for VMGMT-12102

For VMGMT-12102, the terms segments and markers will often be referred to. When a user clicks on a content, it will appear in the preview tab on the right and below the preview there are four tabs: Data, Segments, Markers and QC Markers (see Figure 3).

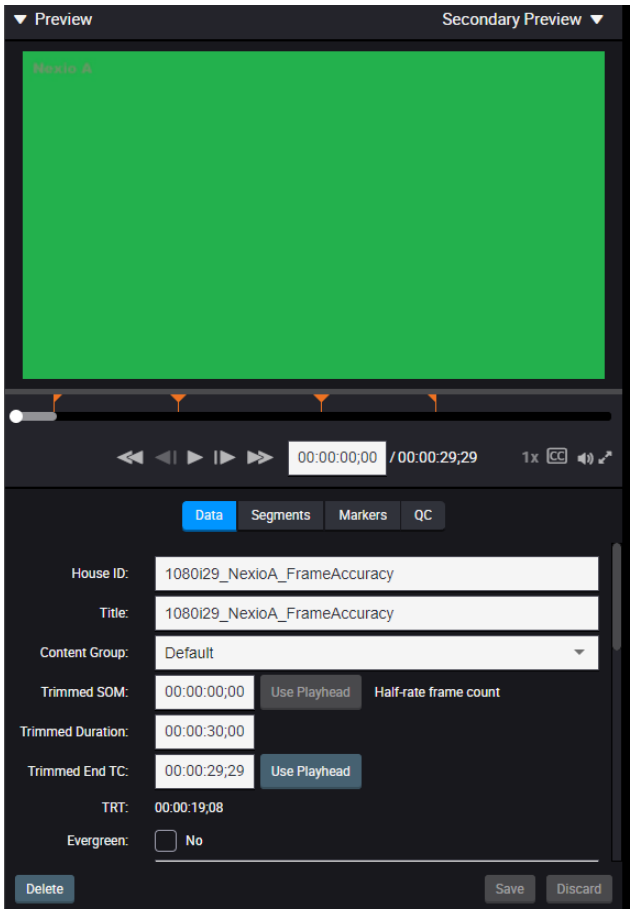


Figure 3: Preview tab of a content

Data is where all the information related to the content is kept such as start time, duration, file size and name, air date etc.

Segments are for breaking down the content into smaller sections or 'segments'. Each individual segment can be added to a schedule in Versio Automation which means that different clips from a variety of contents can be combined. This could involve taking a segment of an advertisement and placing it within a larger clip of a television show.

Markers are used for marking certain spots of the clip but does not have a specific purpose. It is up to the client to decide what the markers mean and how they want to use them. The section QC Markers which stand for Quality Control Markers, are for when the client wants to verify that their content is up to standard to play live. They can mark if a content passes quality control or not and also use the QC Markers to mark where the content fails and assign tags such as "black frame" or "video breakup". Content that failed QC are disabled from being added to a schedule in Versio Automation.

Within the technical analysis, functions such as the select() and moveTo() function are mentioned and briefly described. For a better understanding, the select() function takes in a marker type object and will put it into focus. In other words, in the list of segments, the selected one is highlighted in blue. To find a marker to pass into the select() function, a getMarkers() function is often used. This function returns a list of all current markers or segments depending on the tab, and can be indexed to select them individually. By creating a variable such as markerList that holds the return value of getMarkers(), it can then be indexed; for example, markerList[0] for the first marker, or markerList[1] for the second. This will be useful later when a markerIndex property is introduced. Its purpose is to index the markerList to retrieve the requested marker. The moveTo() function takes in a frame number and moves the play head to that frame. The play head is the white circle that indicates which point of the preview the user it at.

1.5 VMGMT-11963: Important Information

For VMGMT-11963, the case focuses around running automated tests. When automated tests are ran, parameters are given in the command that decide which tests will be run. For example, to only run authenticated tests, the parameter authOnly can be used. For this case, the student will be looking to add parameters for the four different types of users: administrator, operator, superuser and viewer. When the command is run, many functions will execute that determine which test suites are pulled in. A test suite is a group of tests, and each test file will call one of the functions that determine if the test suite should be added to this round of testing. The functions are as follows and each test file calls upon one of these functions to determine if the test suite will run.

- ContentPortalMultipleTests
- ContentPortalAdministratorTests
- ContentPortalOperatorTests
- ContentPortalSuperUserTests
- ContentPortalViewerTests

An enumeration type or enum for short is the name for a set of values. UserTypeEnum was created to hold the set of user types and so all user type comparisons and values will use this enumerated type. For example, the enum UserTypeEnum.ADMINISTRATOR will hold the string ‘administrator’.

2.0 Problem Definition

There are two main problems that the student aimed to solve over the course of the project. This section will detail both problems including what exactly the case is referring to, what the current implementation or workaround is, and why the team decided it needed to be worked on.

2.1 VMGMT-12102

The first case the student worked on was VMGMT-12102: Add the CTRL + ← and CTRL + → shortcuts to the proxy player. Within Content Portal, a user can select any one of their contents and choose to add segments, markers, or QC markers (see Figure 4).

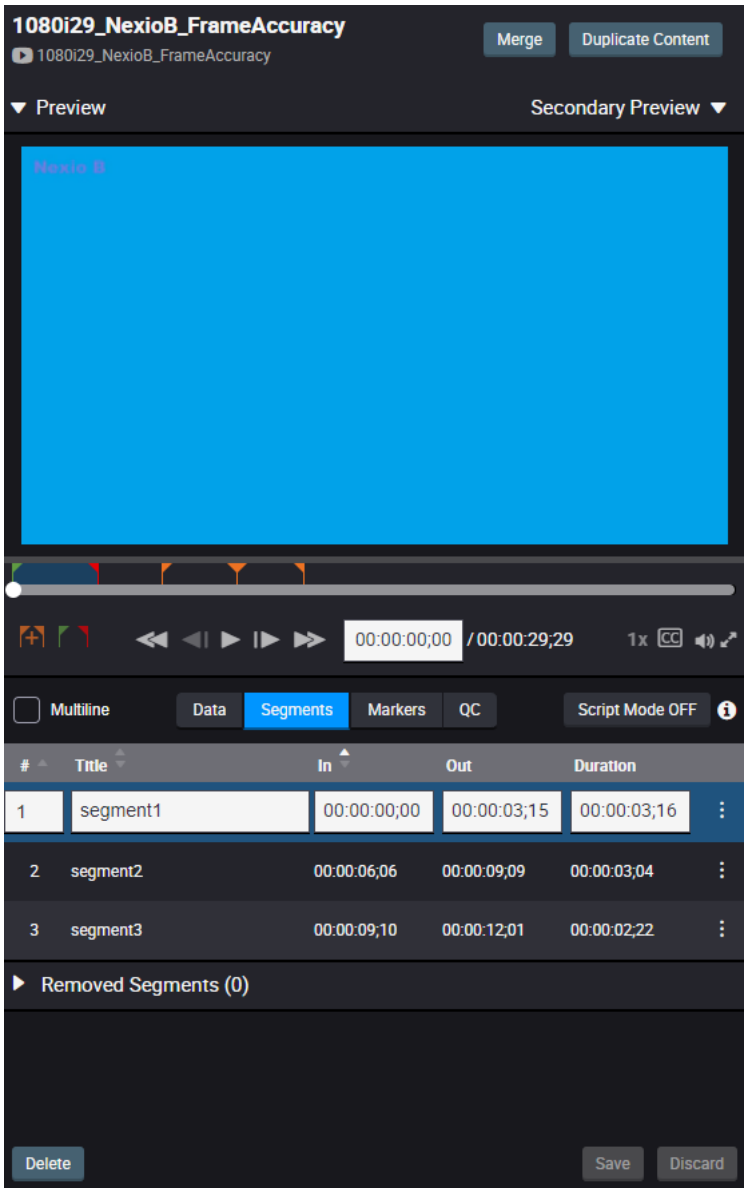


Figure 4: Segment tab of a content

Segments and markers can be added by selecting the appropriate tab, then clicking the orange '+' icon left side. The red and green flags can then be used to determine the start and end boundaries of each segment or marker by clicking on certain sections of the progress bar.

Currently, if a user wishes to access the point at which they set the flag, they will have to click on the flag. While this may not seem like much of an issue, there are certain cases where users will have over 20 segments many of which are within tenths of a second or even completely overlapping. Thus, the team has decided that it would be useful to add a feature that allowed the user to jump between flags with a keyboard shortcut.

The CTRL + ← shortcut will jump to the closest flag to the left, and the CTRL + → shortcut will jump to the closest flag to the right. Another important criterion was that if the user uses the right shortcut while on the rightmost flag, it should jump back to the leftmost flag, and vice versa for using the left shortcut on the leftmost flag.

2.2 VMGMT-11963

The second case the student worked on was VMGMT-11963: Update CP automated tests to use domain users. Before new releases come out, the team does many tests for Content Portal to make sure that none of the changes accidentally affected a previously working feature.

To make the process more efficient, automated tests were written to do most of the testing. An important note is that automated tests are also accessible to other teams who were not involved in the creation of the tests. For a user to access Content Portal, a login is required. Based on the account, a user can be one of four different types of users: an administrator, operator, superuser, or viewer; administrators have the most permissions and viewer having the least.

Currently, all tests that require authentication have been hardcoded to log in with relevant permissions. If a test needs administrator permission, it will log in with hardcoded administrator credentials. In the case that another team does not have the hardcoded account types set up, they will not be able to run the authenticated tests as the logins will all fail. To avoid forcing teams to set up the hardcoded accounts, the Dropframers decided parameters should be passed in with login information when running authenticated tests. This way another team who has their own accounts created can use them for testing and furthermore, can test newly made accounts to make sure their permissions are functioning correctly.

3.0 Technical Progress

3.1 VMGMT-12102 Initial Solution

Shortcuts rely on a file that registers all shortcut combinations, and each shortcut then calls its own function (see Appendix A Figure 11).

The new shortcuts were set up following the same procedure as existing shortcuts and would call the function moveMarker() then pass in a parameter of ‘right’ or ‘left’ depending on which shortcut was used. The moveMarker() function is the new function the student was creating for this case. The discussion that follows relates to the plans and ideas the student came up with for implementing the moveMarker() function.

Progress will be discussed using the term ‘marker’ but refers to all the segments, markers and qc markers tab as their functionality are based on the same functions in the code. The general process for moving the play head to different boundaries would be to find which frame the next boundary is on and the marker it belongs to, then using a select() function and moveTo() function. The select() function is for focusing on the segment, and the moveTo() function is for moving the play head to a specified frame. In Figure 4, segment1 is selected and the play head is at 00:00:00;00.

The student’s initial plan was to first organize all the markers into a list with the leftmost marker being first item and the rightmost marker being last. This was done by analyzing the properties of the marker object and then sorting by the start time property.

The next step was to find what current boundary the user is on, and then match the marker the boundary belongs to a marker in the list described above. Once the marker was found, the student had to determine what the next closest boundary was. This is where the first major problem was encountered.

Each marker object contains both a start time and a duration which is used to determine the end time. This means that just because the markers are sorted by start time, does not mean all their boundaries are in order. For example (see Figure 5), marker A could start at 03:00 and end at 03:10. Marker B could then start at 03:05 and end at 03:08. If the user is currently on the end boundary of marker A, it would be assumed the next boundary is the beginning of marker B but that is not the case. Furthermore, if the user is on the start boundary of marker A, it would be assumed that the next boundary is the end of marker A, but once again, that is not the case.

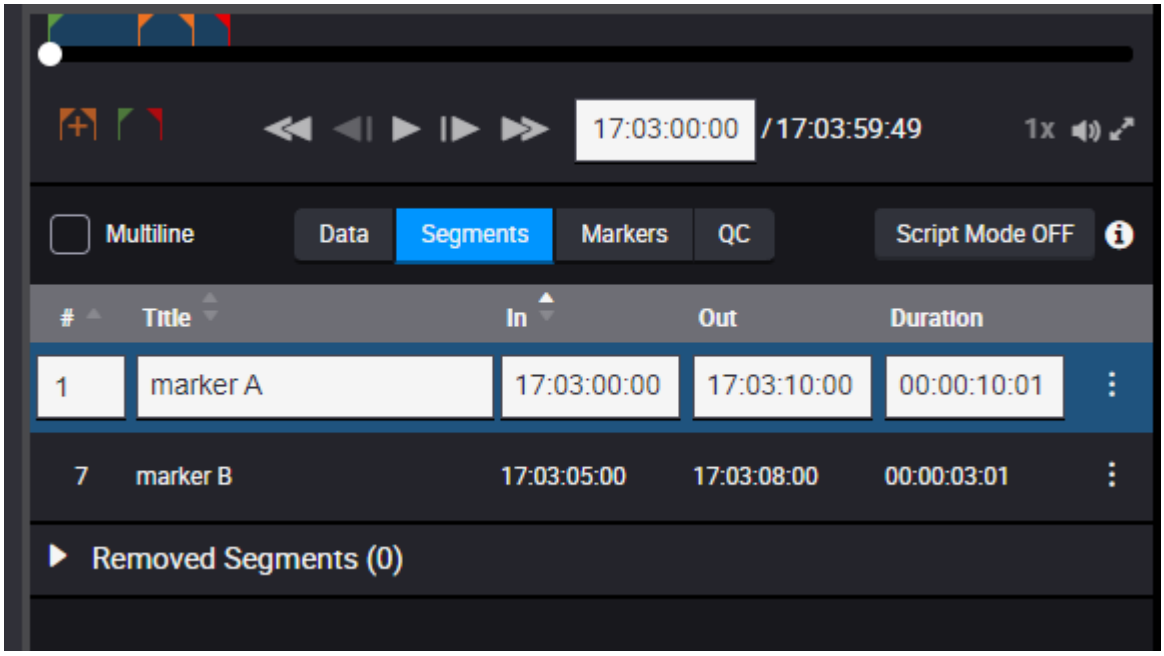


Figure 5: Marker B contained in Marker A

3.2 VMGMT-12102 Problem Solving

The student decided that a list of markers would not suffice as it does not give enough information. The new approach was to use a for loop to go through the unordered list of segments and create a new list called frameList that contained the start and end frame of each segment. This list was then sorted and now the student had an ordered list of each boundary frame. By determining what the current frame the user was on, the student could find the next boundary frame in the list and then match it back to a marker. This solved all cases of nested markers but left one issue with markers that had the same boundary. Since the list only contains frame numbers, there is no way to match a frame back to the marker if two markers share a frame. In this case, a user could infinitely loop through the same two boundaries. For example, if two markers, A and B, shared a boundary on frame 100, then the list would contain the number 100 twice. The current frame would be 100, then when searching through the list, the next closest frame would also be 100. Thus, it would continue to select frame 100 as the next boundary and this would loop forever if the user continued to use the same direction shortcut. This issue is illustrated in Figure 6 where frame 100 is shown twice for visualization.

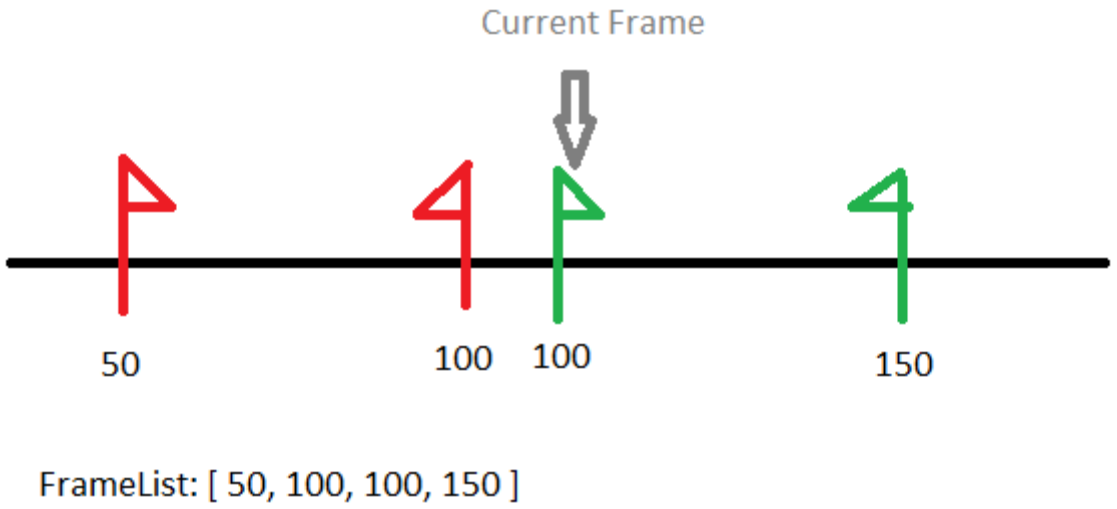


Figure 6: Visualization of matching boundaries

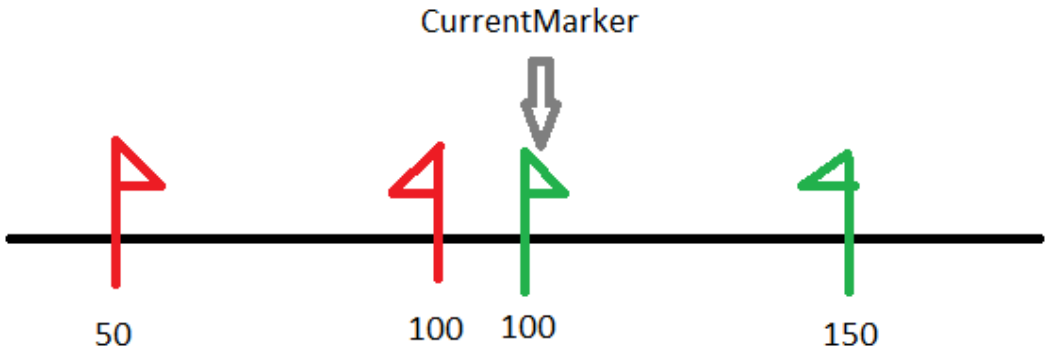
To solve all uncertainties with frame numbers and determining which marker the user was on, the student decided to rework the design of the frame list. Rather than containing all the start and end frames of each marker, it now contains a frame object that has the following information (see Table 1).

Table 1: A table of attributes used for the new frame list

Attribute	Purpose	Sample Value
frameNumber	To keep track of the frame number of the marker boundary	100
markerIndex	A number that maps back to which index the boundary belongs to in the marker list	0; refers to the first marker in the list
markerId	The ID attribute of the marker the boundary comes from to check which marker the frameNumber belongs to	"9a34e501fb7b4493b700aa77a940abca"

3.3 VMGMT-12102 Final Solution

With this new method of organizing the frames, it eliminated the problem of having markers that share a boundary frame. By comparing the id of the selected marker to the id of matched frame, one can tell if that is the correct frame and in turn, also know which frames come before and after. If it does not match, then it will go to the next frame and compare that id until it finds which of the matching frames corresponds to the same marker (see Figure 7).




```
FrameList: [
            { frameNumber:  50, markerIndex: 0, id: abcd1234 },
            { frameNumber: 100, markerIndex: 0, id: abcd1234 },
            { frameNumber: 100, markerIndex: 1, id: efgh5678 },
            { frameNumber: 150, markerIndex: 1, id: efgh5678 }
           ]
```

Figure 7: Visualization with new frame list

With this new method, the student can more accurately determine which frameList item matches the current marker by comparing both the frameNumber and the id. With both properties matching, it would know that the third item is the actual correct frame, and a move right shortcut would go to the last boundary.

Another feature implemented in the final solution was a counter to keep track of which boundary in the list the selected marker matches. This was to account for going past the end boundaries. If the counter was at 0 and the user wanted to move left, it would set the next boundary as the last frame in the list, and if the counter matched the length of the frame list, it would go back to the beginning upon a right shortcut being used.

3.4 VMGMT-12102 Remaining Challenges

After the first review, it was apparent that there were still features missing. Firstly, one must consider the situation where a user is not currently on a marker boundary. In this case, the selected marker's frame will never match one in the list. Furthermore, the play head could also be before the very first boundary or past the very last boundary in which case the current frame will never match the frame of a boundary. Additional features to consider are allowing the shortcuts to be used while the clip is playing, and to make sure that it does not pause a playing clip or play a paused clip.

3.5 VMGMT-11963 Possible Solutions

The student came up with two possible approaches to solve this case. The first option would be to create individual new parameters when the automated tests are run. This means that for each type of user (administrator, operator, superuser, viewer), three parameters would have to be made. A parameter for matching the user type to the enumeration type, a username, and a password. A sample user would look like this:

type: UserTypeEnum.ADMINISTRATOR

username: "admin"

password: "password"

A list of all the required parameters can be seen in the following list.

- adminType
- adminName
- adminPass
- operatorType

- adminType
- adminName
- adminPass
- operatorType

- adminType
- adminName
- adminPass
- operatorType

The second approach would be to create an interface which defines all the components a user would need to enter. An example of the interface in use can be seen in Figure 8 where each user belongs to one parameter with each parameter supplying three properties.

```
{
  usertype: 'administrator',
  username: 'nancytest_admin',
  password: 'Pass123$'
},
{
  usertype: 'operator',
  username: 'nancytest_operator',
  password: 'Pass123$'
},
{
  usertype: 'superuser',
  username: 'nancytest_superuser',
  password: 'Pass123$'
},
{
  usertype: 'viewer',
  username: 'nancytest_viewer',
  password: 'Pass123$'
}
```

Figure 8: Sample Interface values

With this method, only 4 new parameters would have to be added as each parameter can now hold the type, username, and password all within the interface type. The student decided that the second option would be neater and more efficient for a user and proceeded with this implementation.

3.6 VMGMT-11963 Problem Solving

The first issue that arose was that object types cannot be passed in as a parameter. This meant that the chosen method would no longer work, and a workaround had to be found. The student decided that to maintain the tidiness and efficiency of using an interface, the command would now take in properties as individual parameters then place them into an object afterwards.

Additionally, an issue regarding the number of inputted users entered became apparent. Due to the method of combining all the user info into one list, it became difficult to tell how many users were entered. With the user type value hardcoded into the list, the list always had a length of 4 and never contained a null object. Furthermore, when values were left null, the functions that tried to input the log in values got confused as null was not an expected data type. The student came up with multiple ideas that can be seen in Table 2.

Table 2: Possible solutions for null user inputs

Ideas	Pros	Cons
1. Force users to never leave a null value.	No issues with unknown amounts of users or null values being passed into login functions.	Inconvenient for the user if they only wanted to test one account.
2. Allow for null values to be passed into functions.	Anything can be passed in without needing to check the value first.	Does not solve the issue of knowing how many users have been inputted.
3. Add a parameter that gets used when only one user is entered.	Convenient for the user as they only must add one parameter instead of six more.	Does not solve the issue of passing null values into login functions.
4. When a user is left null, return fake values to the login functions.	Easiest method to avoid the null values in login functions.	Does not solve the issue of knowing how many users have been inputted.

3.7 VMGMT-11963 Final Solution

After discussing with the team, the student decided to combine ideas 3 and 4. With two issues needing to be addressed, each idea effectively solved one and thus complimented each other well. A total of nine parameters would now be needed as well as a list to hold all the user type information (see Appendix A Figure 12).

Each user type now has two parameters, a username, and a password. The usertype parameter was removed since it is added into the user info object. The oneUserEntered parameter is required when only one user type wants to be tested. Additionally, a list was created called userList that combined all the parameters to make them more easily accessible. Afterwards, a simple change was needed to replace all hardcoded log in values with the log in values read from the parameters. This required that functions be made to access the parameters (see Appendix A Figure 13).

Similar functions were also made for superusers and viewers. As discussed in idea 4, an else condition had to be added to return a default log in that would always fail due to the login function not accepting null values. In the case where a user type was left out, dummy values are passed in and will fail the log in. However, this will not cause any issues since if the user type was left out, it's corresponding test suites will not be added and therefore, will never run.

The final change was to go into each authenticated test file and adjust the file to call the correct function described in the Introduction. If a test file wants to log in multiple times with different user permissions, the file must call the ContentPortalMultipleTests function. Otherwise, if there is only one log in, it should call its respective function.

Implementing these changes, the new authenticated test works by the user inputting either one or four user parameters. It should be noted that users are informed to only enter one user or all the users and nothing in between. After the parameters are inputted, each test file calls its respective function to see if its tests should be added to the current report. The tests are then run, and the results can be found in an html file. A sample of a one user test can be seen in Figure 9 and an all-user test in Figure 10.

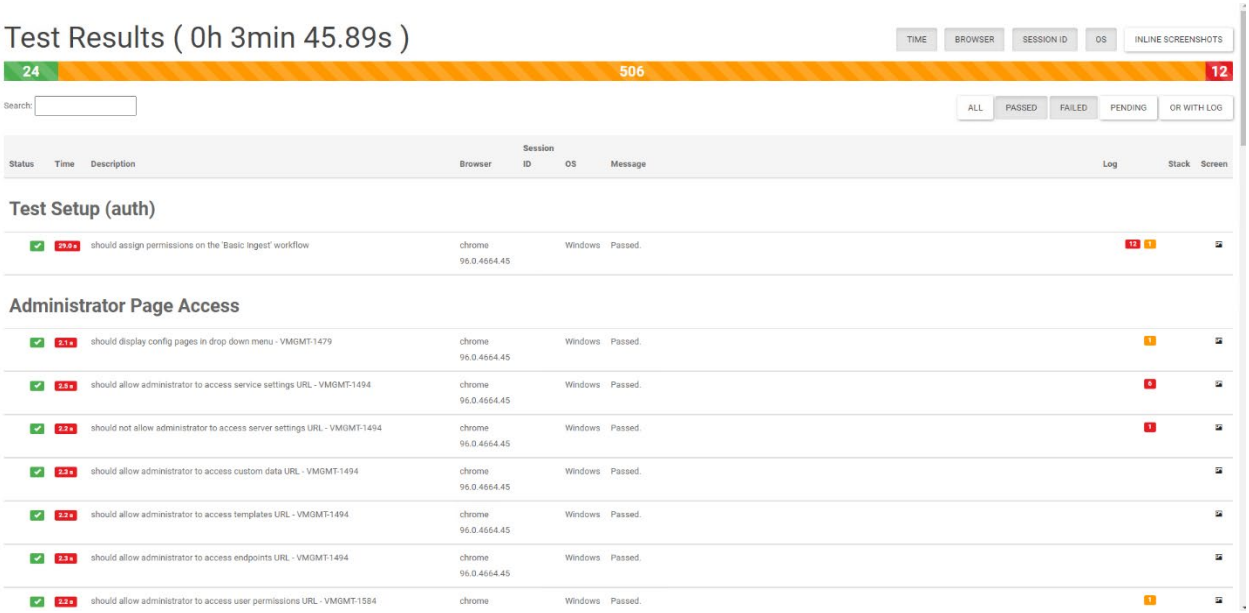


Figure 9: Administrator only authenticated tests

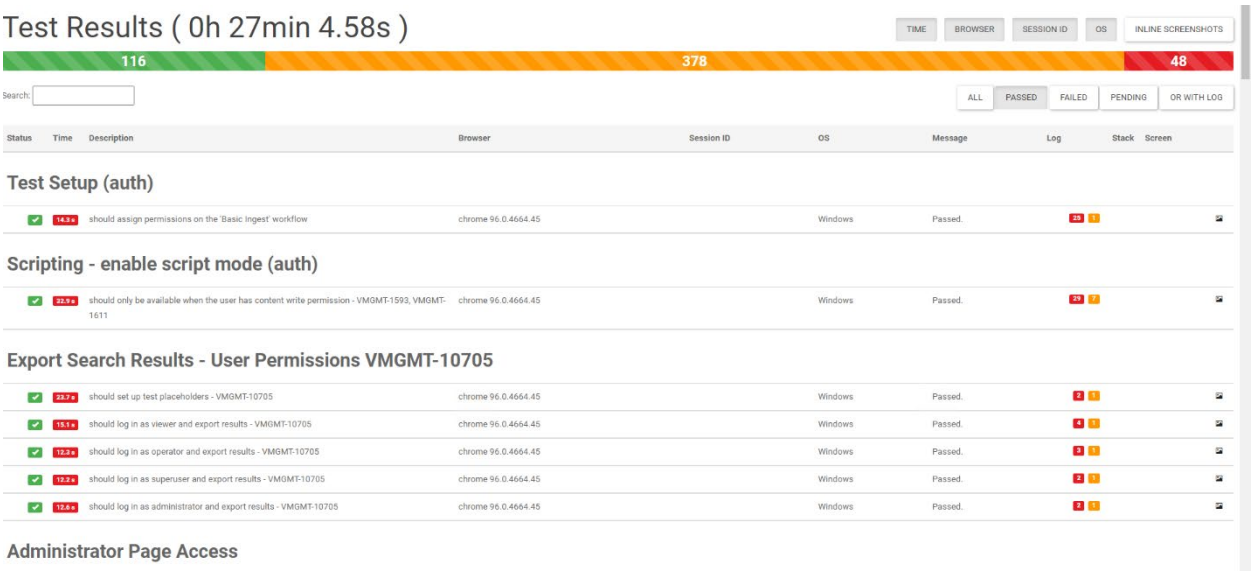


Figure 10: All user type authenticated tests

As seen in the figures above, most of the tests are in the orange bar which means pending. This is because those are the non-authenticated tests and aren't being run. For the administrator only tests, there are much fewer that passed (green) and failed (red) because very few tests are being run compared to the four-user test.

3.8 VMGMT-11963 Remaining Challenges

As stated in the final solution, authenticated tests can only be done individually, or altogether. Another feature that should be added is the option to run tests for two or three types of users. Furthermore, the report file that the user reviews show a lot of pending tests that are not relevant. A better solution would be to remove all non-completed tests from the report, so it does not clutter up the important results. Finally, a test suite can have one authenticated test and many non-authenticated tests, but because of the one authenticated test the entire suite will be added in. The test files should be refactored so that test suites are organized into authenticated and non-authenticated groupings to keep the non-authenticated tests that run to a minimum.

4.0 Conclusions

4.1 Conclusions

To reiterate, the student was tasked with solving two cases to implement new features for Versio Content Portal. Those features were shortcuts for moving between segments and domain users for automated tests. The 4.7 GA has not been released yet and so there are no quantifiable statistics regarding the impact the new features had on customers or other employees of Imagine Communications.

Regarding the shortcut case, the student successfully implemented a first iteration of segment shortcuts. The shortcuts were designed by having a list of frame objects to precisely find which segment came next and then moving the play head accordingly. This solution allowed for basic functionality of segment shortcuts but there was still room for improvement regarding using shortcuts while a preview is playing.

For the second case, the student also succeeded in adding domain users. The student added new parameters to the testing commands that allowed for the user to input user logins and passwords and based on the amount and type of users entered, it would run the appropriate tests. This solution met the team's expectations, but future cases would like to investigate adding more customizability to the parameters and more relevant data in the report file.

4.2 Next Steps / Recommendations

For future members who want to build off the work the student has completed, there are quite a few areas where improvements can be made. For improving upon the shortcuts, one should consider adding the option to use the shortcut while a preview is playing and when the play head is outside the boundaries of the first and last segment boundary. Another improvement that could be made is finding a more efficient way to determine the next closest boundary. Currently a while loop is used to find which boundary the current frame is at and so if there are a multitude of segments, it could slow down the efficiency.

Regarding next steps for the domain users, it has been mentioned that there is currently a handicap that only allows for one or four users to be entered. This method does not provide the user with the best experience and so a member should investigate adjusting the conditions of the test functions to consider two or three users being entered. Furthermore, the report the tester reviews after the tests are complete is cluttered with pending test cases. All tests that weren't required to run based on the parameters should not appear in the report. In addition to reorganizing the report, the test suites should be refactored to only contain closely related tests so additional tests won't run when they are not required.

References

[1] Imagine Communications, "Versio Content Portal," Imagine Communications, [Online]. Available: https://imaginecommunications.com/product/versio-content-portal/. [Accessed 15 December 2021].

[2] ProductPlan, "General Availability (GA)," ProductPlan, [Online]. Available: https://www.productplan.com/glossary/general-availability/. [Accessed 16 December 2021].

[3] J. Academia, "Lecturing Technical Writing to 1st Year Engineering Students," *Journal of Engineering Education,* p. 6, 2017.

[4] F. Tipster, "An Analysis of the Impact of Teaching Methods on Student Academic Performance," in *Engineering Education Conference*, Seattle, 2015.

[5] APA, Referencing in APA Format, Waterloo, 2010.

Appendix A

```
private bindKeys = () => {
    this.shortcutService.bindKeys([
        { keys: ["delete"], command: this.deleteSelectedMarker },
        { keys: ["ctrl", "c"], command: this.copySelected },
        { keys: ["ctrl", "v"], command: () => { this.pasteMarkers(); this.$scope.$digest(); } },
        { keys: ["ctrl", "left"], command: () => { this.moveMarker("left"); } },
        { keys: ["ctrl", "right"], command: () => { this.moveMarker("right"); } },
    ], KeyScope.preview);
}
```

Figure 11: Shortcut set up and function calls

```
public getAdministrator = () => {
    if (this.userList[0])
        return this.userList[0];
    else
        return { usertype: UserTypeEnum.ADMINISTRATOR, username: "admin", password: "password" };
}

public getOperator = () => {
    if (this.userList[1])
        return this.userList[1];
    else
        return { usertype: UserTypeEnum.OPERATOR, username: "operator", password: "password" };
}
```

Figure 12: Functions that return user parameter values

```
private adminName: string = browser.params.adminName || null;
private adminPass: string = browser.params.adminPass || null;
private operatorName: string = browser.params.operatorName || null;
private operatorPass: string = browser.params.operatorPass || null;
private superuserName: string = browser.params.superuserName || null;
private superuserPass: string = browser.params.superuserPass || null;
private viewerName: string = browser.params.viewerName || null;
private viewerPass: string = browser.params.viewerPass || null;
private oneUserEntered: boolean = !!browser.params.oneUserEntered;

private userList: (IUserInfo | null)[] = [
    { usertype: UserTypeEnum.ADMINISTRATOR, username: this.adminName, password: this.adminPass },
    { usertype: UserTypeEnum.OPERATOR, username: this.operatorName, password: this.operatorPass },
    { usertype: UserTypeEnum.SUPERUSER, username: this.superuserName, password: this.superuserPass },
    { usertype: UserTypeEnum.VIEWER, username: this.viewerName, password: this.viewerPass }
]
```

Figure 13: Properties for user types