Video on the Web

System Design Document

Kyle Schulz, Aaron Hart, John Langewisch, Andy Niccolai
Colorado School of Mines - Field Session 2011

ABSTRACT

Video has become a prominent component of the Internet. As such, it is critical that the web management system Web Crescendo, a product of Silicon Mountain Technologies, be able to offer such services. These services must be closely integrated into the existing system in a manner both secure and simple for users. In order to meet this need, a video streaming service has been created and integrated into the Web Crescendo application. This service contains a streaming media server, a suite of client tools for video integration, a security protocol that allows or restricts video access on a per-user basis and a video library mechanism for video management.

Table of Contents

Introduction

Silicon Mountain Technologies (SMT) is an e-Business Partner that provides a variety of web services including a product called Web Crescendo (WC). WC is a web management platform that enables the layman to quickly build and modify very robust and complex websites.

WC previously did not support the uploading and displaying of videos on these websites short of manual implementation, defeating the purpose of an "approachable by anyone" model with powerful results. This also means that WC had no way of securing videos (i.e., allowing only certain users to access certain videos).

In light of the growing demand for video services, SMT had asked our team to develop a system for incorporating videos into the current WC system as well as create a service to authenticate users when they attempt to view such videos. The system we developed allows clients to include videos in their websites using WC and provide security with which clients can specify who can and cannot view those videos.

Previous System Architecture

The current system allows users to log in to the WC service from the user's browser. When a web page hosted by WC is requested, WC goes through a checklist to determine whether the user is allowed to visit that particular web page. Prior to the video server, WC handled all HTML traffic. There were no other servers that a browser communicated with when viewing any web page.

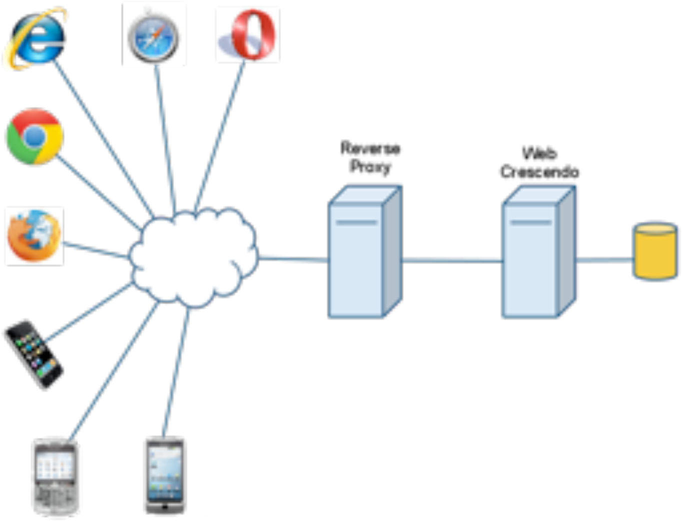


Figure 1 - Old Architecture

First WC decides whether the website for that page is secure. If so, WC then determines if the user requesting the page is logged in and if that user has access to that particular website. Then WC does the same authentication for the particular web page, determining whether the web page is secure, whether the user is logged in, and if the user has access to that particular web page.

Creating Web Pages

Users can log in to the WC service and create web pages from "portlets." These portlets are essentially pieces of a web page (a title, a form, an image, text, etc.). To create a web page the user simply chooses which portlets to include in the web page in what order. WC is essentially a content management system (CMS) with some added features. The user can also view and edit the source HTML. However, using the current WC service the only way to add a video to a web page is to reference an external video not hosted by SMT.

Requirements

The goal of the project was to integrate a video streaming service into the WC system. The first four functional requirements are necessary to provide basic streaming capabilities and were a top priority, whereas the last three requirements were secondary goals that would have provided extra functionality on top of the basic streaming services.

Functional Requirements

1. Stream video to all major browsers and devices.
2. Enforce role access constraints.
3. Expose functionality to administrators as a module inside WC.
4. Video playback controls should look similar to YouTube.
5. Allow clients to stream live video to all clients.
6. Videos are searchable with embedded metadata.
7. Videos are viewable in a library view inside of WC.

Non-Functional Requirements

1. Choose which video format(s) to support.
2. Choose a Streaming Server with appropriate capabilities for this project.

Design

The four main components of the streaming video system are the client interface, security mechanism, video encoder and video library.

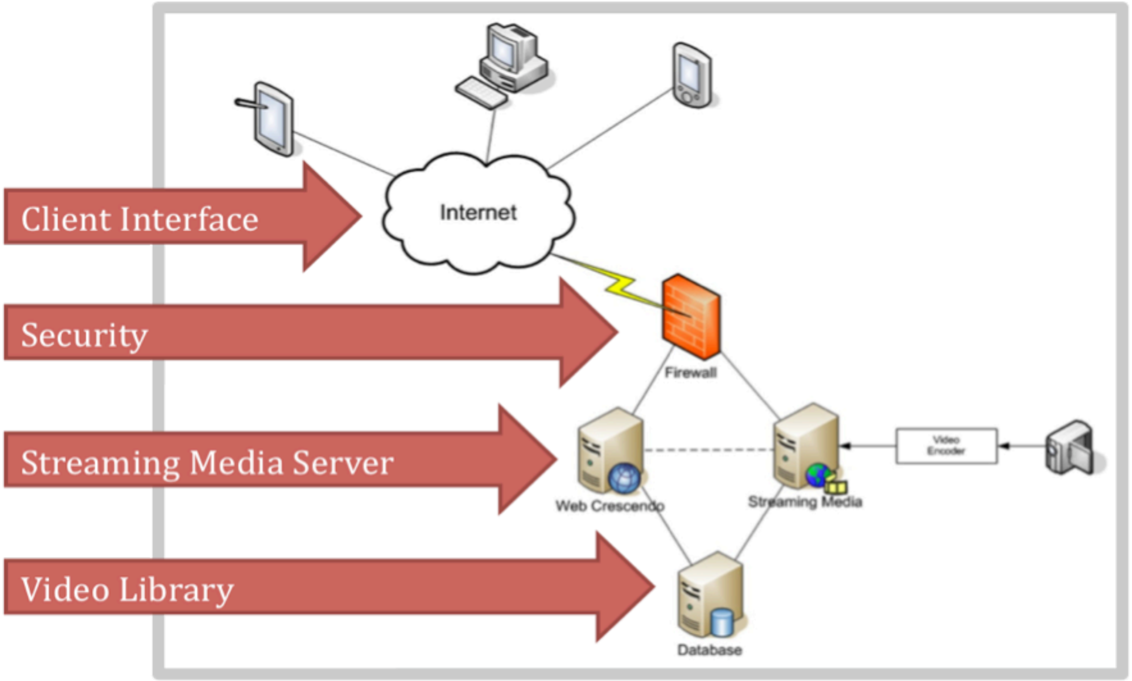


Figure 2 - System Scope Overview

The client interface component handles what a user sees and interacts with when adding video, as well as the underlying mechanics of embedding that video onto a web page. The security component determines who can and cannot view a requested video and handles video dispatch. The video library component organizes videos, making them available to both the client and the other components of the system. Lastly, the video encoder component (labeled in figure 2 as the "streaming media server") converts client submitted video into formats dictated by the system standard.

Component: Security

Security (or more appropriately "User Authorization") is an essential component to this project. User Authorization allows clients to control who can and cannot view videos. This provides clients with a video hosting service that is both powerful and secure than other video hosting services such as YouTube.

Security - The Sequence of Events

All videos are streamed from a server running Wowza Media Server 2 software. For the purposes of this document this server will be referred to as “Wowza”. If a video is placed on a webpage, then when a user visits that webpage the user’s browser sends a request for the video to Wowza as shown at the top of figure 2. This video request includes the URL of the requested video and the user’s Session ID. The Session ID is created when the user visits a website hosted by Web Crescendo and is stored in the user’s browser cookies.

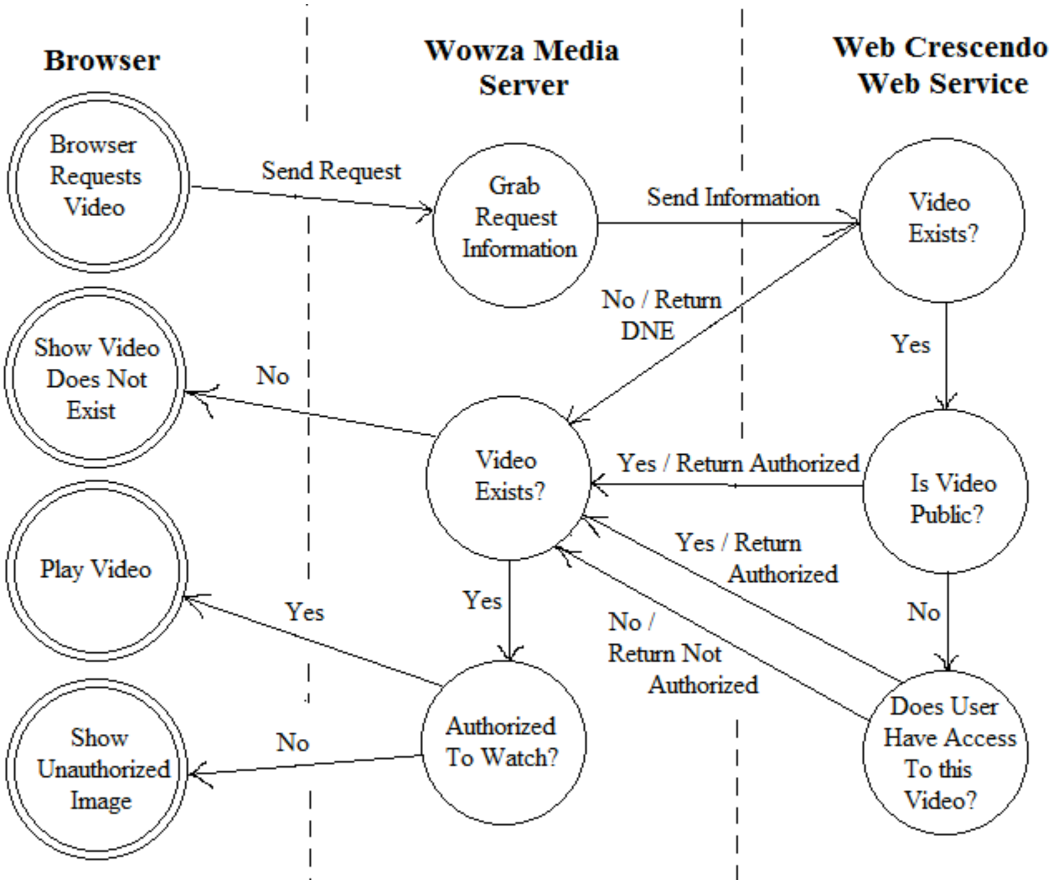


Figure 3 - Finite State Machine for the Authorization Process

Before Wowza locates the requested video it retrieves the information from the request. Wowza passes that information on to a Web Service hosted by the Web Crescendo Server. Using this information, the Web Service determines whether the user is authorized to view the requested video.

To determine whether the user is authorized or not the Web Service compares a user role to a list of video roles. The user role and video roles are grabbed from a request and compared to ensure that the user has the correct roles to view the video. Once the Web Service has determined whether the user is authorized to view the video it returns an appropriate response to Wowza.

After Wowza receives a response from the Web Service it separates out the data in the response and decides whether or not to play the video. This effectively prevents unauthorized users from viewing non-public videos.

Security - Retrieving User Information

The first step in the process is to halt the video streaming and retrieve the video URL and Session ID from the video request. To do this a servlet is placed on Wowza to listen for video requests. This servlet is responsible for grabbing information from the video request. Since there are several different protocols used to request videos (RTMP, HTTP, RTSP) the servlet determines what protocol is being used to request the video and retrieves the information from the request appropriately. After the information is obtained and formatted correctly an authorization request is made to the Web Service.

Security - The Web Service

The next crucial step in the process is the Web Service. The Web Service is also a servlet but is located on the Web Crescendo Server. This servlet is responsible for using the provided information in the request from Wowza to determine whether the user is authorized to view the requested video.

Authorization on the Web Service is done using a user role and video roles to determine authorization. To get the user role for the request the Web Service uses the JSESSIONID from the request to get the session information for the client. This session information contains a variable for the user role. For the video roles the Web Service uses the video URL in the request to retrieve the video roles for the specified videos by querying the Web Crescendo database. Once the Web Service has the user role and a list of video roles it simply checks to see whether the video roles contain the user role. If the video roles do not contain the user role then the user is not authorized to view the requested video.

Once the Web Service determines authorization it forms a response and sends that response back to Wowza.

Security - Establishing the Connection

An HTTP connection is used for communication between Wowza and the Web Crescendo Server. Using an agreed upon port, the two servers are able to communicate reliably and securely. SMT libraries are used to create a Connection Manager on Wowza that is able to send HTTP requests to the Web Service on Web Crescendo. The Web Service requires information from these requests in order to do authorization. The Web Service responds to Wowza with a custom XML file which contains an authorization code, an error, code, and an error message. This ensures that information is passed correctly and that communication between the servers can be modified or extended as needed.

Security - Determining Whether to Stream

After requesting authorization from the Web Service, Wowza listens for a response. After receiving the response Wowza will decode the XML file into its components. If the authorization code is true then Wowza starts streaming the video to the client. If not, then Wowza checks the error code and error message to determine why the user was not authorized and if an error might have occurred.

Component: Video Library

The video library is responsible storing and managing video data. Specifically, it manages video metadata and video permission roles, as well as any information that might facilitate searching for videos. The video library adds a set of tables to the WC database. Two JavaServer Pages (JSPs) are used for managing and selecting videos. A WC component (Action) defines the behavior of the video management portlet (a piece of web content) in the JSPs.

Video Library - Web Crescendo Integration

The WC application enables users to drop in various portlets and easily create their web page without having any knowledge of web design or coding.

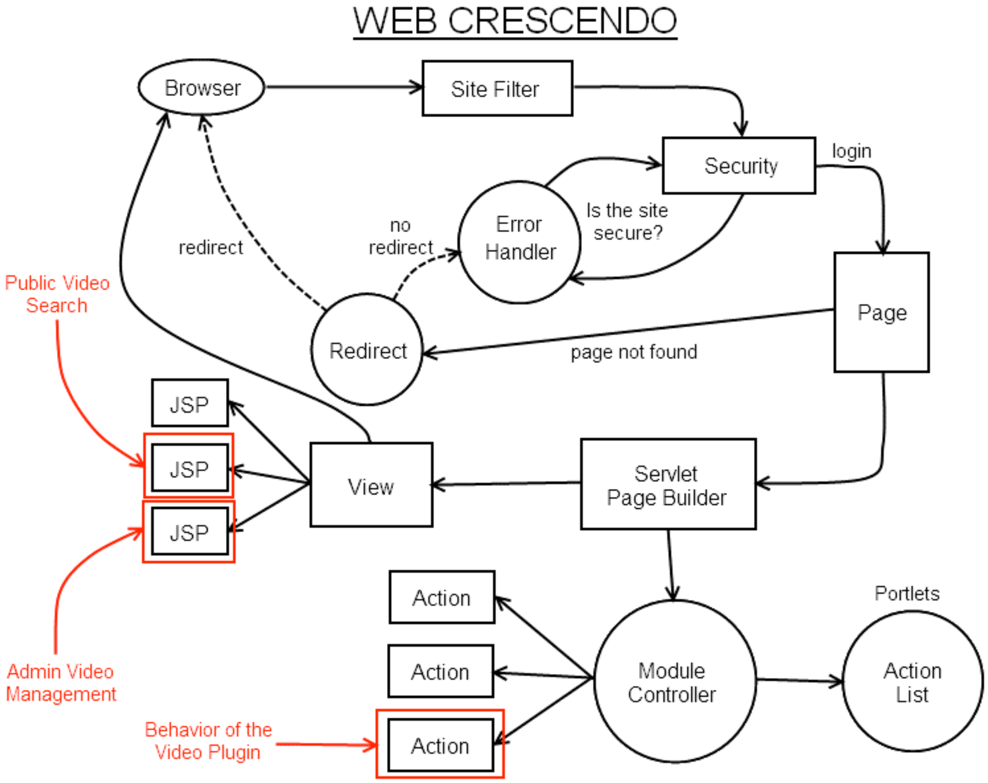
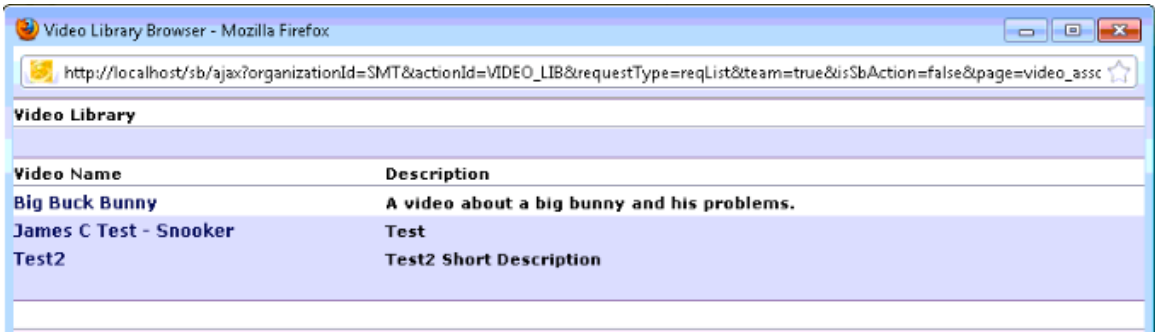


Figure 4 - Video Library Integration with Web Crescendo

A client logs in to the Web Site Command Center to access the WC tool and the "Page Builder Servlet." This is controlled by the "Module Controller," which goes through the list of portlets (defined by Actions) that the client has put onto their page. It then inserts the appropriate HTML to define the behavior for each portlet. The Page Builder Servlet also has JSP defined "Views" that lay out the portlets on the page. Each portlet potentially contains multiple Views. These Views define where the component will be positioned on the page (e.g. a content action can be placed in the first column, second column, etc.). Figure 4 illustrates these interactions.

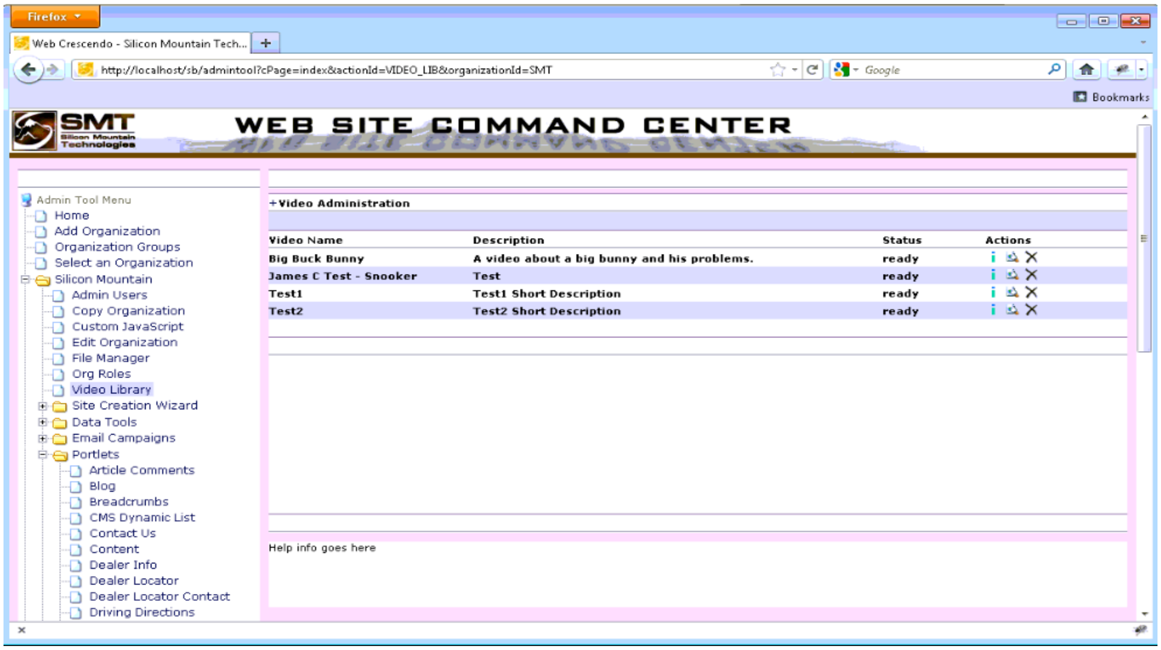

Video Library Browser - Mozilla Firefox
http://localhost/sb/ajax?organizationId=SMT&actionId=VIDEO_LIB&requestType=reqList&team=true&isSbAction=false&page=video_assc

Video Library

Video Name	Description
Big Buck Bunny	A video about a big bunny and his problems.
James C Test - Snooker	Test
Test2	Test2 Short Description

Figure 5 - Video Library Browser

The first JSP lists the videos that are available to be put in a piece of content. It is called when the client is creating a piece of content and selects the “add video” option from the editor. A list of videos will pop up, allowing the client to chose and place them into their web page, provided they have appropriate user access roles and the video is available to be played. The list is generated by querying the database for all the videos that are both under a client’s control and available to be played from the Wowza media server. The client can click on any of the videos in the list and a URL for that video will be returned to the client interface. A view of the video list is shown in figure 5 above.



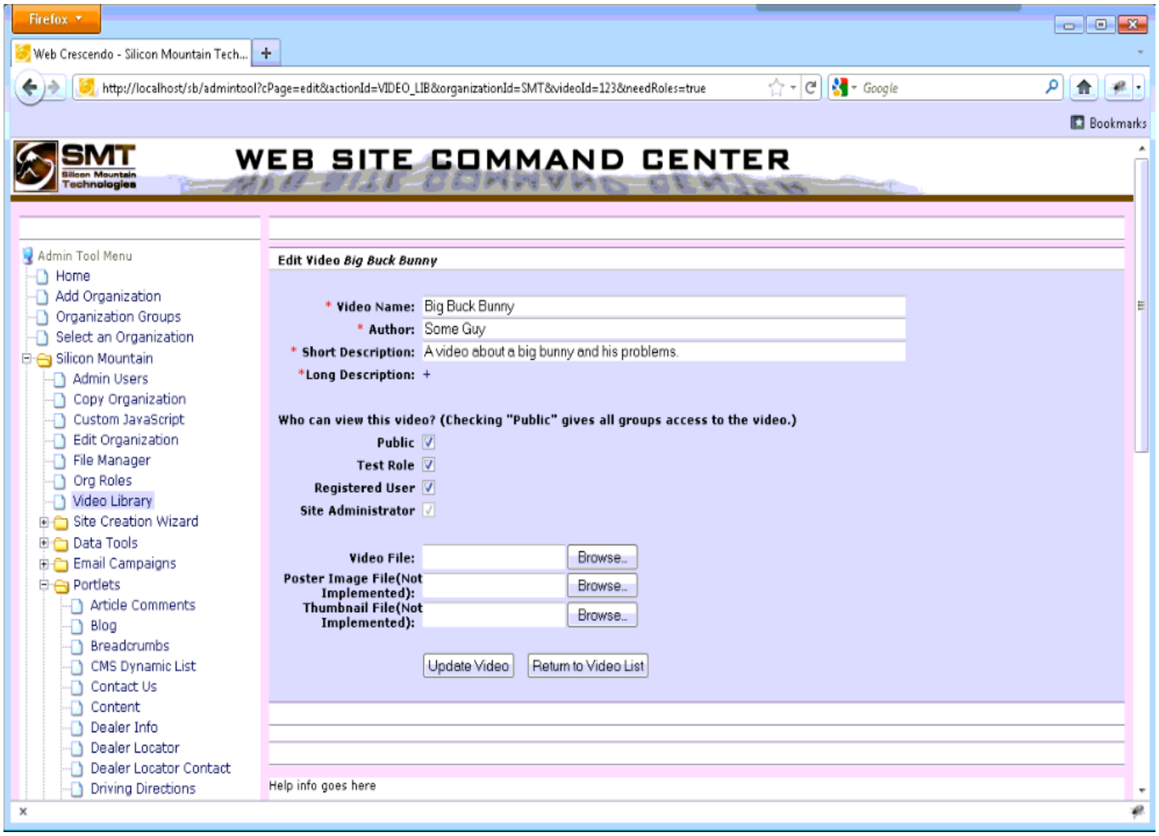
+Video Administration

Video Name	Description	Status	Actions
Big Buck Bunny	A video about a big bunny and his problems.	ready	i ✎ ✕
James C Test - Snooker	Test	ready	i ✎ ✕
Test1	Test1 Short Description	ready	i ✎ ✕
Test2	Test2 Short Description	ready	i ✎ ✕

Help info goes here

Figure 6 - Video Management

The second JSP contains the layout for the video management tool (see figure 6). The tool allows the client to add, update, and delete videos. To add a new video, a client selects the “add” icon, which brings up the dialog shown in figure 7. The dialog box includes fields for various video attributes. It also allows the client to set the video permissions. Finally, the dialog box allows the client to upload the video file to WC.



http://localhost/sb/admintool?cPage=edit&actionId=VIDEO_LIB&organizationId=SMT&videoId=123&needRoles=true

Edit Video *Big Buck Bunny*

* Video Name: Big Buck Bunny
* Author: Some Guy
* Short Description: A video about a big bunny and his problems.
*Long Description: +

Who can view this video? (Checking "Public" gives all groups access to the video.)

Public ☑
Test Role ☑
Registered User ☑
Site Administrator ☑

Video File:
Poster Image File(Not Implemented):
Thumbnail File(Not Implemented):

Help info goes here

Figure 7 - Video Upload/Update

The JSPs do not define their own behavior, which is instead controlled by a WC Action. This Action includes methods for connecting to the database, retrieving the list of stored videos, listing the videos for placement on a piece of content, and allowing the client to create, update, and delete those videos. It also contains an API that can be used by other WC components.

Video Library – API

In order for the video library component to integrate easily with the other components, an easy-to-use API was created that allows access to important video data. The API includes functions to access and set video attributes including permission roles, any physical attributes of the video such as dimensions, file size, or length of the video, and any other pertinent information that the other components of the video portlet might require.

Specifically, a function called "getVideoRoleIds(String videoUrl)" was created to return a list of the video roles that a video currently has. The security portion of this system uses this for video authentication in order to compare the user's role to that of the video to ensure they are able to watch the video.

Video Library – Database Schema

The video library management system adds functionality to the current WC database. Metadata for each of the videos uploaded must be stored allowing for easy access to provide information to other components in the system.

VIDEO:

- Video Id
- Video Name
- Author
- Create Date
- Update Date
- Video URL (location of the video file on the media server)
- Running Time
- File Type
- File Size
- Default Aspect Ratio Height
- Default Aspect Ratio Width
- Thumbnail URL
- Poster Image URL
- Short Description
- Long Description

VIDEO_ROLE_XR:
(Cross reference table for videos and video roles)

- Video role Id
- Video Id
- Role Id
- Create Date

It should be noted that this only includes the part of the planned schema that was actually used in the final implementation. There are also tables in the database for the support of a video tagging and filtering system as well.

A request to the database returns all of the pertinent information for a certain video or videos. The reason for returning all of the data instead of just a certain field is that it is much faster for a database transaction to return all of the data in one row and have the application parse that data later, rather than having multiple database requests be made just to retrieve certain fields.

In addition to the schema requirements, the implementation of the additional database tables is also constructed such that the tables will "self-clean" when files are created, deleted, or updated. This will ensure that there is no inconsistent data in the database.

Component: Client Interface

The video player interface has several controls to allow viewers to interact with the video. These controls are used by most video players and expected by users.

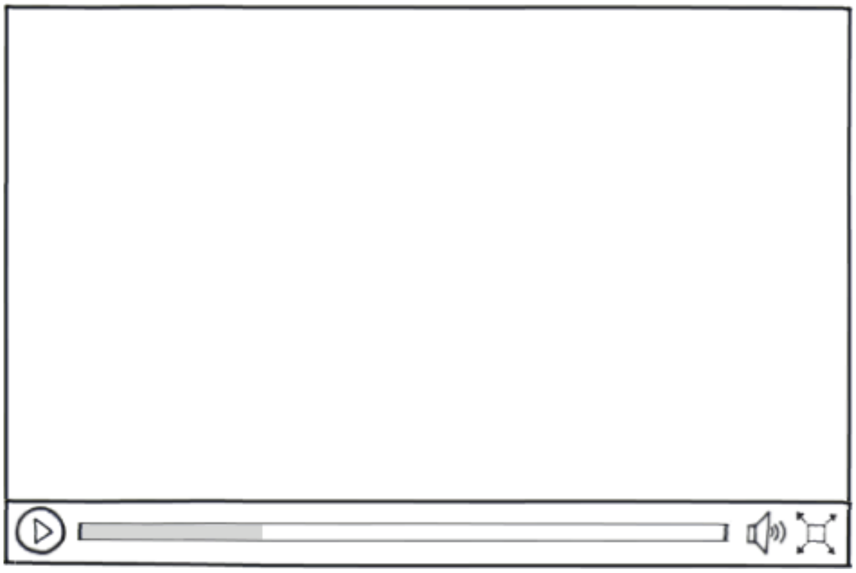


Figure 8 - Video Player Controls

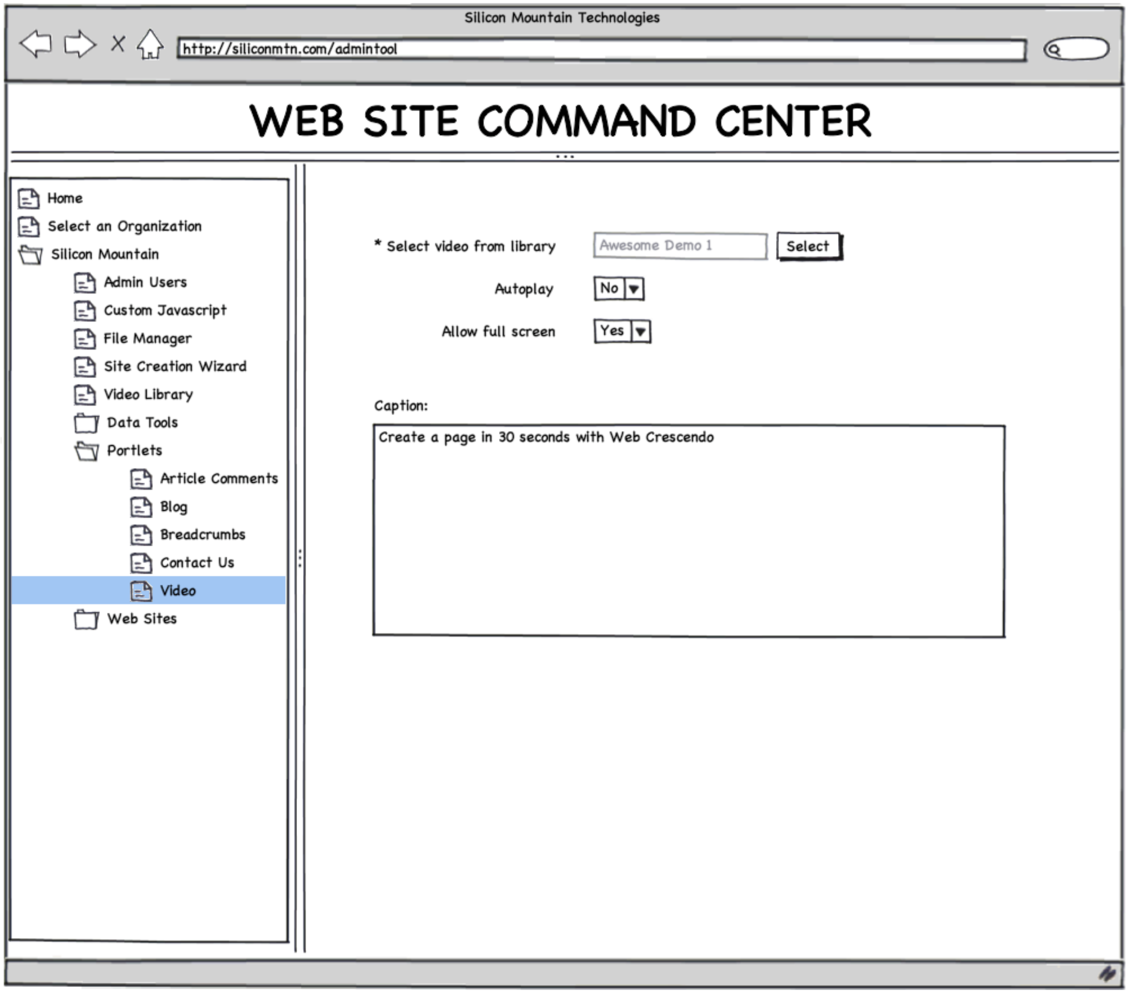
These controls are illustrated in figure 8 and perform the following functions:

- **Play** - Starts the video or continues playing if the player is in the paused state.
- **Pause** - Stops the video if playing while maintaining the position on the seek bar.
- **Seek Bar** - Provides visual feed back on how much of the video the user has watched and allows the user to go to any part of the video by clicking on the bar.
- **Volume** - Changes the player volume without affecting the user's system volume and brings up a slider upon clicking.
- **Full Screen** - Allows the video to fill the entire screen during playback.

If autoplay has been enabled, the video begins playing when the user visits the page. Otherwise, the video player initially displays a static image if an image has been provided. Since the first frame of a video is usually black, this feature allows users to visually recognize the video. The user will see an error message if they are not authorized to view the video.

Client Interface - Administrator Controls

Web Crescendo allows administrators to add content such as images and text to web pages through the WYSIWYG HTML editor.



* Select video from library: Awesome Demo 1

Autoplay: No

Allow full screen: Yes

Caption:

Create a page in 30 seconds with Web Crescendo

Figure 9 - Video Settings from within Web Crescendo

Administrators can add video to a page through the video dialog. The video dialog in figure 9 will have the following controls to allow administrators to embed videos:

- **Browse Videos** - The user must choose a video from the library to be played.
- **Title** - The name of the video
- **Width/Height** - The dimensions of the video.
- **Begin Playing Immediately** - If "Yes" is selected the video will begin playing as soon as a user loads the page. Otherwise, users must click on the video to being playback.

Component: Video Encoding

In order to both maintain a consistent service and support client needs, some standard of media must be enforced in the form of the video's codec, container and format. A model must be established for the conversion of client submitted video to this standard in both form and function.

Video Encoding - Background

A codec is simply an encryption of a data stream. In the media setting, a codec also handles data compression, allowing naturally large files of visual and/or audible data to be significantly reduced without a dramatic impact on quality.

A container is a file wrapper that contains meta-information describing the wrapped data, including the location and usage of specific pieces of information. For a video file this includes items such as frame rate, chapters, key frames and the utilized compression codec.

The format of a video file refers to the actual presentational elements, namely the bit depth, frame rate, bit rate and aspect ratio. Bit depth describes the color information and may not need any enforced limitations at all. Frame rate determines how many frames are displayed per second, and while the Internet carries a standard of 12-24 fps (frames per second), videos seldom exceed 30 fps. In extreme situations, the absolute maximum would be 60 fps as anything higher is unperceivable by the human eye. Bit rate is discussed in the following paragraph and aspect ratio ensures that oddly sized videos conform to a standard that clients can easily implement into their web sites.

Bit rate is the single most important format to control as it directly translates to both user experience and server performance. Described as the rate of transfer of information, a high bit rate means larger amounts of video information per frame. While this means a better quality video, it also places a higher demand on the server and network connection. Tests will have to be run to determine the ideal bit rate for our client's needs, but initial estimates place it at around 300 kb/s. A future solution could include a dynamic bit rate encoder that looks at average server load and serves an appropriate media file.

Video Encoding - Conversion

Conversion of video comes in two major forms: re-encoding and multiplexing (more commonly referred to as muxing). Re-encoding is a long and intense process of evaluating the frames and re-compressing them using a different codec. Multiplexing is the chaining of streaming information together and is used in the video realm to "embed" one video into the container and codec of another blank video, allowing a wide range of formats to be interpreted as one. Due to its popularity, simplicity and speed, our solution made use of muxing.

The H264 codec (typically contained in an mp4 file) was selected to support a broad range of target players (browsers, devices, etc). However, it is unlikely that most (let alone all) of the media uploaded by a client will initially conform to this format. As such, it is necessary that the media be converted to the H264 standard.

There are innumerable video conversion tools available, ranging from simple open source command line muxers to expensive (yet powerful) cloud-driven re-encoding suites. While many were initially considered, the specs of four prominent systems were compared.

Tool	Decoding	h264?	Interface	Speed	Quality	Cost
Zencoder	All	Yes	Cloud	High	High	$.05 / min
FFmpeg	Most	Yes	Command	High	Med	None
Jave	Most	Yes	Java	High	Med	None
Adobe	Some	Yes	Application	High	High	None

Figure 10 - Encoder Comparison

When selecting a conversion service it is important to consider how many formats are supported as a valid input. All but a few supports the mainstream formats (avi, ogg, mov, mpg, etc.) but only the expensive, commercial applications boast a 99.8% support rate. However, due to the integration methods described later, an application or service-based converter does not fit with the established server model. An additional consideration is that of client confidentiality, ruling out any cloud-deployed applications.

All considered converters supported the need H264 codec as well as a high encoding speed (due to either powerfully backed servers or muxing). Given its flexibility and integration capabilities (as well as the strong recommendations of Wowza developers), the series of libraries by the name FFmpeg, developed by the media community to handle nearly every format in existence, was selected.

FFmpeg is command line driven interface and can be compiled on any system needed, allowing flexibility in both integration and hardware selection. However, depending on the method of integration, it may be greatly beneficial to support a Java-based interaction. For this purpose, the Jave package is listed as a potential candidate, which simply wraps the FFmpeg suite in a Java environment.

Video Encoding - Potential Work Flows

Two potential integrations were considered for the actual integration of the video conversion process. One converts on the fly through a Java interaction and the other converts through a schedule using the command line.

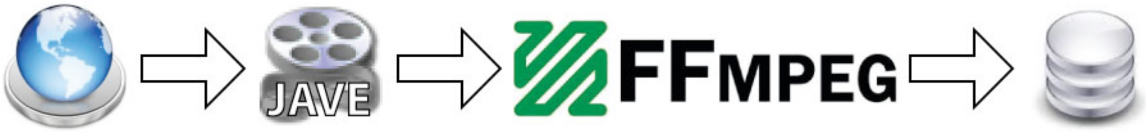


Figure 11 - Encoding Workflow A

Figure 11 shows the first approach. A client uploads a video that eventually gets sent to the Jave interface. Jave does some quick analysis on the video and runs an FFmpeg script on it. The FFmpeg script and Jave continue to communicate, providing the client with feedback on the conversion process. Once completed, the video is either rejected (with errors) or approved and saved to the final location.

Video conversion is a very expensive process, even when simply muxing. Due to this fact and that of FFmpeg's location ON the media server itself, it is unwise to have both streaming and converting occurring simultaneously.

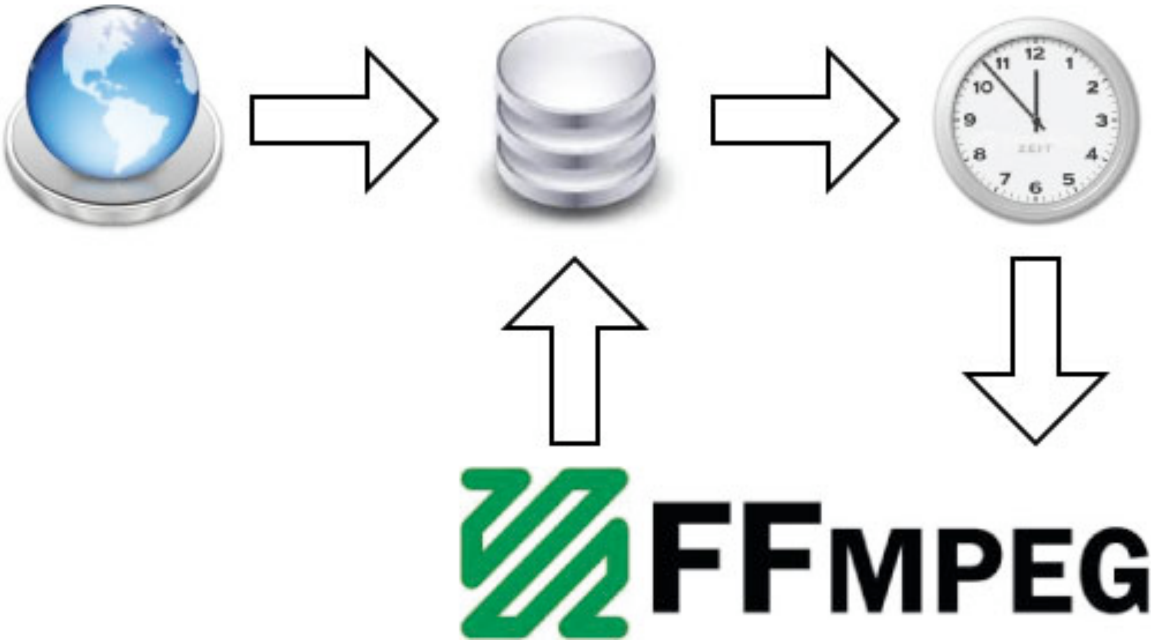


Figure 12 - Encoding Workflow B

Figure 12 illustrates one option (also the chosen method) to address this concern. By running FFmpeg as a scheduled process, the taxing conversion process could be performed in the late hours of the night while little to no video is being served. A client would upload a file into a designated "pending" folder and await conversion. Upon the appointed time, FFmpeg will fire up and automatically convert all videos in this pending state.

In a client model, such delays are both expected and unfortunate. Tools could be put into place that skip this process and go live immediately if the current format is already acceptable, or a separate server could be tasked with handling the conversions.

Video Encoding - Handbrake

The need to manually build FFmpeg with little to no guidance or support and track down all relevant libraries (as is the nature of most open source projects) led to the decision of replacing FFmpeg with the Handbrake command line interface (CLI). Handbrake is an application built around the functionality of FFmpeg and as such inherits every aspect of its compatibility and efficiency. Yet in the event where new formats are needed or updates are released, a simple update of the Handbrake application replaces countless hours of trying to re-build FFmpeg.

Implementation

Wowza

There are two methods for passing information through a web request. The first is through cookies, which are only passed through HTTP requests. The second is query string, which appends information to the URI and can be used for all web requests. Since Wowza receives requests using HTTP, RTMP, and RTSP, query strings were chosen to pass information.

Wowza needs two pieces of critical information to perform authentication: the stream name and JSESSIONID. The stream name is the unique URL of the requested video and is used by the web service to get the video's roles (denoting permissions). The JSESSIONID is used to identify which client is requesting the video and thereby obtain the client's user role. This allows the web service to compare the video's roles with the user role for authorization.

Web Service Security

Though authorization by the Web Service is a relatively simple procedure, some important details must be considered. As there are several reasons why authorization might fail, it is important to also include them in the response to Wowza (in the form of an error code and message). For example: if the database connection failed, Wowza may receive an error code "1" and message "the network is currently own." Wowza then logs this information.

As WC is not hosted on one machine, but many, a configuration file for Wowza is necessary to obtain correct paths and ports. Furthermore, the JSESSIONID of a user is specific to a particular WC server and configuration is needed to keep track of which servers contain the relevant session.

The Video Library

Java Database Connectivity (JDBC) was used for all database connections and queries, due to its ease of use and the format already in being implemented by the client's current system. In order to facilitate the interaction between the database and the system, value object classes were constructed to parse all of the database query results. This presented an easy-to-use solution to the object-relational mismatch that often is encountered in dealing with database-backed applications.

A value object class was also created for the video roles. A video value object contains a map of video role value objects (corresponding to all of the available roles to be set for that video). This video role structure is used in the upload/update JSPs to dynamically create the appropriate role checkboxes.

When a new file is uploaded, the video stored under a new name consisting of a unique id ensuring there are no conflicts. The video is then stored on the Wowza server under the same name in a file structure consisting of the Wowza base video directory and the organization id. This URL is stored in the database so that when the encoder has finished processing a video, it can easily move the file to the appropriate location.

The Client Interface

Though the interface a client sees is written in HTML, JavaScript, and Adobe Flash, the interface a video viewer sees is dependent upon the context they're viewing the video in. After the browser has loaded a page containing video, one of three players will be inserted into the page by client-side JavaScript.

1. **Wireframe Placeholder** - Static element that shows the size and poster image of the video.
2. **HTML5 Video** - HTML element embedded with a simple tag.
3. **Flash Video Player** - Flexible, but requires the browser to have Flash installed. Not supported by iOS.

The WYSIWYG editor initially displays placeholder, as the browser cannot automatically execute the JavaScript. Next, while loading the video, the browser replaces the placeholder with the HTML 5 video element. If Flash is supported on the device the Flash player replaces the HTML5 video element. This order allows the correct player to be loaded on a device.

Strobe Media Player is Adobe's default flash media player. The video attributes are passed to Flash through a JavaScript variable. A JavaScript library will only embed the video if Flash is supported, allowing mobile devices to use the HTML5 player instead.

In order for Wowza to authorize users, the JSESSIONID is read from the browser's cookies and appended to the end of the video URL via query string. JQuery is called to perform most of this functionality, which masks most browser's incompatibilities and includes functions to perform common tasks. The cookie library for JQuery is used in the client interface and allows the client-side code to grab the JSESSIONID.

Administrator Controls

CKEditor is a free JavaScript library for building toolbars that allows users to edit documents in Microsoft World like environment. Web Crescendo uses these toolbars to allow site administrators to add content such as text and images, and now video, to pages. The video button inserts the entire HTML needed by a page to display a video.

Scalability

As a specific requirement (as well as a general good practice), each component must be decoupled from the machine it operates on. This mainly allows for ease of scalability, as moving to a new machine (or many machines) will not break your code.

The main method of achieving this was through configuration files. For example: instead of hard coding the location and functionality of loading information from a database, simply place the login credentials and address of the database in a config file. Upon initialization, the code will load this config file and place it into an arguments map to be accessed by the rest of the application. If the application is deployed on a new machine, or several machines, the config file drives the connectivity instead of the code.

When combined with certain network environments this also allows for dynamic processing where the config information is passed in on the fly (rather than loaded from a file), allowing for on-demand allocation of resources. Your code could be deployed on a random server and receive jobs or requests from any and every source to be sent to any and every destination, all without having to change the code or re-deploy your application.

Console Commands in Java

The process of wrapping a console application in Java introduces a significant and considerably under-documented problem, involving deadlock. When a console command or application is run, the output and error streams of the currently running operating system handle various feedbacks. This may be in the form of a status update (output) or an unexpected format (error).

When Java runs a console application, these streams must be processed in some manner. Typically a while loop is run on a collector that pulls info from the stream, line by line, finishing once the stream is empty. This works well in concept and especially well in simple environments, as simply having a while loop for the output stream and then another while loop for the error stream should display all messages and exit correctly.

The problem arises due to the fact that while one stream is full the other is blocked. If a message is waiting in the error stream, it will not allow processing on the output stream. If the application makes use of the error stream and the output stream intermittently, two consecutive while loops will cause deadlock as your output loop sees an output stream with info in it, yet the presence of info in the error stream prevents it's access.

One must concurrently process the streams to bypass this problem. By setting up each stream reader in it's own thread, no deadlock will prevent one from reading as the other will eventually clear.

Results

In general, this project was a complete success. No functional requirements were unable to be addressed and every problem that arose ended with an elegant solution. Each component was designed to scale to multiple machines without having to be re-written and integrated seamlessly with the existing WC system.

System Testing

Tests were established for each of the four components as well as integration tests for the entire system. In addition to acceptance tests, simulated server load tests will be performed to test capabilities under stress.

System Testing - Acceptance Tests

Security

- *Can anyone view public videos?*

  Public videos are viewable by anyone, regardless of a JSESSIONID existing or not.

- *Can permitted users (and only permitted users) view secured videos?*

  If a user requests a video but does not have permission to view it, the request is rejected. Likewise, if a video is requested anonymously (i.e. a user is not logged in) and is not public, the request is rejected.

- *Will a change in user roles correctly block a previously viewable video?*

  To test this requirement, a user with sufficient privileges opened a page with an embedded video. While viewing the video, the roles of the user were changed to remove permissions. Upon refreshing the page, the user could no longer receive the video. Likewise, changing the video role from public to private also denied further video requests.

Video Library

- *Can administrators upload videos to the video server?*

Administrators are able to utilize the uploading utility to correctly upload and update videos.

- *Can administrators set role-based permissions on individual videos?*

  Administrators can associate roles with individual videos using the video management tool.

- *Can administrators set the title, size, and poster image for uploaded videos?*

  Unfortunately poster image is not currently supported due to time constraints. Title is able to be customized and size it inherited from the file itself.

Client Interface

- *Can administrators select videos from the video library?*

  Administrators can choose a video from the Browse Videos dialog while inserting a video. The video parameters are automatically passed to the editor and can then be inserted into the document.

- *Can administrators set the video to start playing when the page is loaded?*

  Autoplay works on desktop browsers, but is disabled by Apple on iOS devices. Administrators can enable autoplay for desktop browsers by checking the box labeled 'Begin playing immediately' while inserting a video.

- *Does the video player select the correct video stream for desktop browsers?*

  The Flash player is not supported on iOS devices so the JavaScript does not embed a Flash element after it embeds the HTML5 video element.

Video Encryption

- *Are encoded videos playable with Adobe Flash?*

  As Flash is the default player for video served by the Wowza platform, all encoded videos must be (and have been tested to be) compatible with Flash.

- *Are encoded videos playable on the iOS emulator/iPhone/iPad/Android?*

  Tests for the iPad and iPhone have allowed video to be played. An emulator for the Android has yet to be tested.

- *Can the encryption script run multiple files?*

  The encoder is built to run any number of jobs as needed and correctly runs them sequentially. Due to the nature of the batched process, no concurrent execution is needed.

- *Can the encryption script handle very large/long files?*

  Tests were run on larger, high definition files without error. An instance occurred where a video with custom surround-sound tracks did not have it's audio correctly converted, yet this was considered inconsequential as such audio tracks are rare and only present in very specific DVD and BluRay rips.

- *Are odd aspect ratios supported?*

  Non-standard aspect ratios are correctly converted without error and their respective database entries updated to reflect these properties.

Integration

Integration testing of the components was necessary to ensure that a working result was delivered to the client. The code was deployed to a preproduction web server, running the WC software with our additions and tested for functionality. Other machines housed the Wowza server and the encoding application to allow the testing of cross-resource communication.

Conclusion

Client Review

A full demonstration of the finished project was given to the client. Using a computer independent to the project, the client was able to successfully upload a video from his computer, have the video encoded correctly and have the video put on Wowza. The client was then able to create a new webpage on Web Crescendo and include the video as part of the page. The client was able to view the video from the webpage when the video was given public permissions, but correctly had to log in to the web crescendo service in order to see the video when the permissions for the video were changed to non-public.

Lessons Learned

SMT has numerous base libraries for common tasks like parsing query strings, creating database connections and making HTTP requests to servers. Before writing any generic functionality it is important to check the base libraries for existing resources. These resources are already incorporated into the Web

Crescendo platform and already take care of various configurations and errors that can occur.

The JavaServer Pages Standard Tag Library (JSTL) is especially useful to dynamically create lists or parameters for videos. In addition, simple logic may be placed into the JSPs that allow for more dynamic content control. However, JSPs should not contain any "business logic" and simply display content in the proper format.

Despite initial impressions, "open source" does not translate to "viable commercial solution". In a business environment where software stability and support is crucial, the conglomerate of code representing most open source applications created more problems than it solved. In the case of FFmpeg, building a version that included the H264 support required an involved process of tracking down libraries and troubleshooting builds, relying on information posted in various forums dating as old as five to ten years. In the event encoding needs changed, a new build must be made with new libraries and generally required much more technical expertise than should be needed for such a simple implementation. For these reasons we recommend using a live, updating commercial application that is backed by customer support... you really do get what you pay for!

When creating methods & classes it is important to not only include documentation surrounding the method (like Javadocs) but also to include comments inside each method identifying what the method is doing. This allows someone else to view your code as an overview rather than needing to decipher each line of code to figure out what is happening.

It is also important to make sure that methods do one task and one task only. By creating methods that only do one task you create functions that others can use without needing the context of your application. For example, one of the functions in the Web Service is getVideoRoles(). This function does one task, which is getting the video roles. It does not get video roles AND compare them to a role ID, etc. This allows someone else who needs video roles to use this function without needing the Web Service application. Furthermore, you will find that when you separate individual tasks out into their own methods the bigger more important methods become rather small and very easy to read.

Remember that exceptions are your friends and that you should use them as needed. This is especially true if you are correctly creating helper functions, which are context free. For example, if getVideoRoles() had a NullPointerException because the video couldn't be found getVideoRoles() should throw that exception instead of consuming it. This is because getVideoRoles() doesn't (and shouldn't) know what needs to be done in the event of a NullPointerException. However, remember that exceptions should only be used for non-standard program flow. If a method returns null but you process that null immediately then it is better to just check the return value for null rather than attempting to do something with the return value and catching a NullPointerException.

You should always comment which development kit version you are using when importing libraries. For example, include a "JDK 6.0" above your imports from the JDK 6.0. Remember that just like every piece of code libraries are living and continually change. Utility functions will act differently in different versions. When attempting to update or debug your code it is important to know what version of libraries are/were being used.

Build scripts will save you a lot of hassle. When developing code for servers it is a regular task to create archive files and redeploy those files to a test server. In our environment this included creating .jars of our classes and exporting them to specific destinations on our test machines. Doing this by hand would take considerable time. However, using build scripts Eclipse can be configured to re-jar and export selected files every time you perform certain actions, like saving from your project! What this meant was that for developing on the Wowza Server updating the server with the correct jars was a simple task of saving the project.

Future Directions

Support for video poster image files and thumbnail files had to be left out due to time constraints. These would need to be implemented in the future to provide easy video library navigation and custom video denial messages.

Live streaming video was a potential feature that did not become realized due to time constraints. Not much research was devoted to its functionality as a result, but Wowza does support it by default and in theory should not be too difficult to implement.

The database schema currently shows support for a tagging and filter system. Due to time constraints and an analysis of the scope of the project, this portion of the design was unable to be completed. However, this element of the video library should be able to be added easily. It would have a very similar structure to the video roles. Modification will need to be made to the video library Action in order to support this as well and the JSP page for uploading and updating a video will need to also be updated to support the adding of tags to a video.

Glossary

API	Application Programming Interface
CLI	Command Line Interface
CMS	Content Management System
Handbrake	An open source, GPL-Licensed video transcoder
H264	A modern standard of video encoding classically contained in an MP4 file structure.
HTML	Hyper Text Markup Language (refers to the standard format used to create web pages)
JDBC	Java Database Connectivity, a Java API that enables Java programs to execute SQL statements
JSP	JavaServer Pages
JSTL	JavaServer Pages Standard Tag Library, encapsulates as simple tags the core functionality common to many Web applications
MP4	A video container format that describes meta information.
Portlet	A modular section of web content
SMT	Silicon Mountain Technologies, Inc. (our client)
VM	Virtual Machine
WC	Web Crescendo (the current web application of SMT)

References

Project Requirements - see attached.
Original project requirements document as supplied by the client.

Handbrake - http://handbrake.fr/
Homepage for the Handbrake encoding utility

Handbrake CLI - https://trac.handbrake.fr/wiki/CLIGuide
Info on the usage of Handbrake's CLI

Wowza - See listed below
General reference and information on the Wowza media server

http://www.wowzamedia.com/video-streaming-server.html
http://www.wowzamedia.com/forums/
http://www.wowzamedia.com/resources/WowzaMediaServer_UsersGuide.pdf
http://www.wowzamedia.com/resources/WowzaMediaServer_ServerSideAPI.pdf

Silicon Mountain Technologies

Web Crescendo Application
Streaming Media Services

Table of Contents

Revision History

Date	Name	Description
5/16/2011	James Camire	Creation of Document

Overview

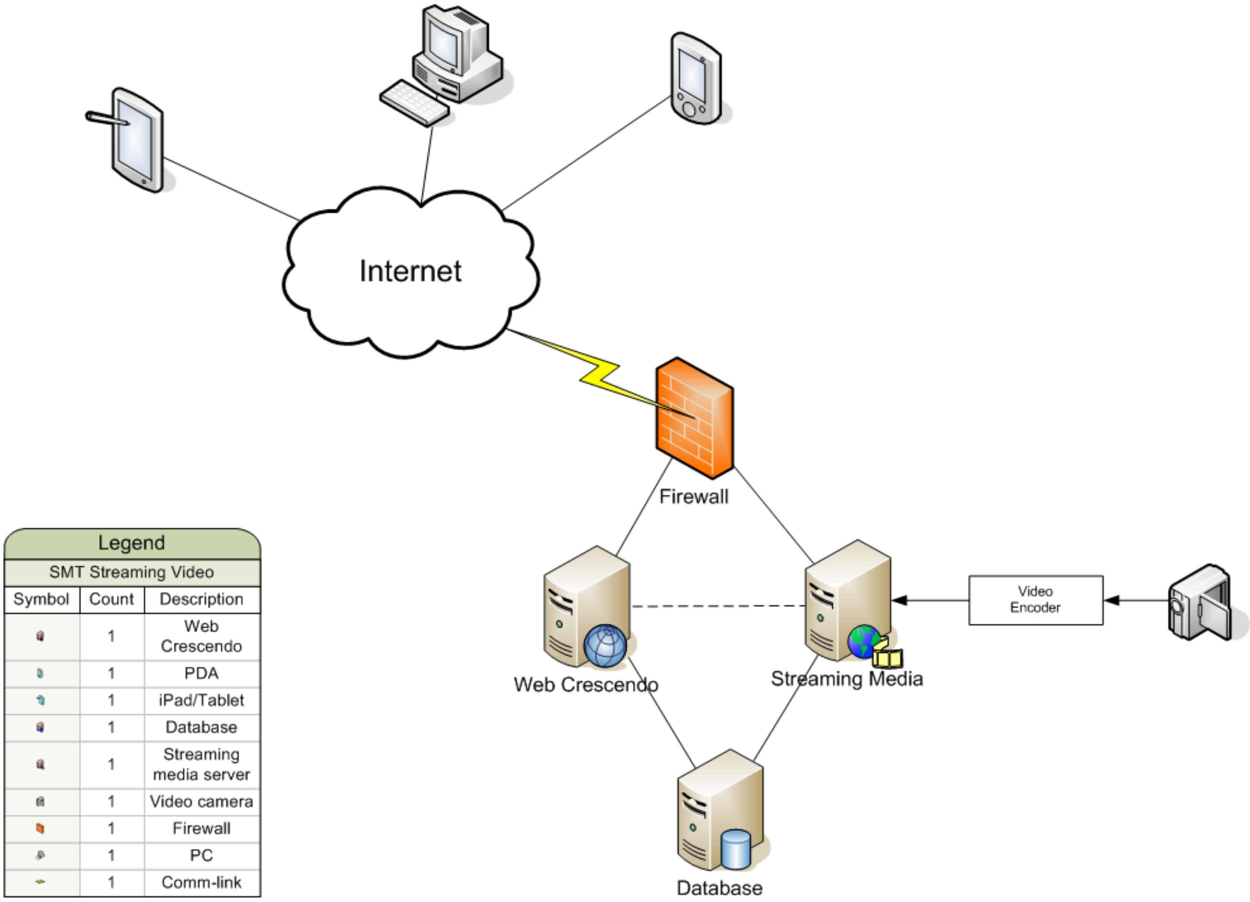
The SMT Web Crescendo platform is a state of the art, Web Management System that allows our customers to quickly and easily manage large, complex web sites. Over the course of the past few years, video services have become a more important component of the standard website offering.

With the launch of the Web Crescendo Learning Management Platform (Q4 2011), video streaming services will become an important component of the Web Crescendo application. Because video is such an important component of the LMS, users will need the ability to easily integrate video from their video library into a page on the LMS course.

While there are many excellent 3rd party video-streaming services, these offerings all lack a crucial component. Namely, the ability to have a tight integration between the video and the security model implemented on their website. This integration is the most critical component of the service offering.

Scope

This project will be tasked with creating a Video Streaming Service and integrating the service into the Web Crescendo application environment. This project is broken into a number of components. These components are listed below in their order of importance. Sections 2.1 through 2.3 are REQUIRED components for Phase 1 of this project. Figure 1 shows the major components to this project.



Legend		
SMT Streaming Video		
Symbol	Count	Description
	1	Web Crescendo
	1	PDA
	1	iPad/Tablet
	1	Database
	1	Streaming media server
	1	Video camera
	1	Firewall
	1	PC
	1	Comm-link

Figure 1

Server Selection

The selection of the streaming media server is perhaps the most important component of this project. The server, at a minimum, must support the following capabilities:

- Must have a robust API that will allow a developer to implement various features through external code

- Must be able to interrupt the request pipeline before the stream is started and reroute the request to an error handler in case of unauthorized access
- Must be able to easily add new videos to the repository through file managed implementation or the API
- Part of the server evaluation process is the determination of the video formats necessary for delivery to the targeted platforms. These platforms include:
    - HTML 5 compliant mobile devices, with special emphasis on iPhone, Android and Blackberry devices
    - HTML 5 compliant browsers, including Internet Explorer, Firefox, Chrome, Safari and Opera
    - Flash video support for browsers that are non-compliant with HTML 5
- While it is probably not possible to support one video format for all of the supported devices, the goal is to standardize on 2 video formats.
- Since we are a small company, we need to ensure that the chosen server platform has attractive licensing options.

Client Selection / Development

The selection of the client tools necessary to seamlessly display the video is extremely important. It should be easy to include a video client on an HTML page and have the appropriate video/video client displayed. The client should support the following features:

- Standard controls (volume, play, pause, skip, etc ..) should be available and easy to turn on/off
- Client should allow for easy styling/formatting through the use of CSS
- Both HTML 5 and Flash clients must be supported
- Must have the ability to automatically detect the user device and use that for display on that device
- Build a plugin for the CKEditor WYSISYG tool that will allow users to select a video from the library, choose a set of features for the player (size, controls, etc ..) and will insert the appropriate HTML onto the article.

Web Crescendo Security Integration

Once the server selection has taken place, integration between the WC platform and the streaming Server must be implemented. The following sequence of events should occur:

1. Browser requests access to a video.
2. Streaming server will call a WC web service and pass the following information:
   a. Requested Video
   b. URL of the request
   c. User JSESSIONID for the request
3. WC web service will identify the video in the library and retrieve the access information for that video.
4. WC web service will identify the user role based upon the provided JSESSIONID
5. WC web service will compare the user role to the video roles.
6. WC Web Service will return a status code to the streaming server
7. If user is authorized to view the video, video will begin streaming to the user
8. If the user is not authorized, a not authorized image will be returned for display

It is important to note that the domain name for the website and the streaming server must be the same. Otherwise, the JSESSIONID will not be passed. WC uses a cookie path of "/", which means the session id will be passed to any path on the server.

SMT will be implementing a layer 7 load balancer for this project. All videos will be requested by: http://domain/media

Video Library and Encoding

Users must have a mechanism to upload videos, manage their access roles and be able to delete videos in the library. Since a single video format will not allow us to support all desired devices, we will need to encode videos into multiple (hopefully only 2) formats upon upload. The following use case will be implemented for this project:

- Admin screen in the WC admin tool that will display a list of videos.
- Users should be able to edit video meta-data and access controls
- Users should be able to delete videos in the library
- The following sequence should occur when adding a new video:

a. Fill out meta-data
b. Select video from their file system
c. Pre-check to ensure the format is supported by our encoders. We need to define this list of formats
d. Video will be streamed to the server using SMT's File Upload filter.
e. Video will be converted into multiple formats and moved to the appropriate place on the file system
f. Database will be updated to include the pointer to the video as well as the meta-data

- Users should be able to preview their video from the admin tool

Real-time Video Streaming

The WC streaming services must support the capability to perform real-time video streaming through a WC website. While this service will require significant capability from the vent coordinator, there must be a WC process/capability in place to facilitate this offering. This component must have the following capabilities:

- Predefined mechanism stream data from the remote location to our streaming servers
- Real-time delivery through the users website
- Ability to add the stream to a web page before the event via the WC admin tool
- Saving of the stream into the proper formats and adding to the video library for future viewing

Miscellaneous Capabilities

In addition to the core capabilities, we need to also have the ability to do some of the following:

- Add meta-data information to the WC content indexer
- Ability to view page/video after searching
- Optimize the viewers and pages for SEO
- Ability to use a CDN (content delivery network) in the future while also maintaining our security integration points

Conclusion

This service offering must be in line with the other capabilities (LMS, Email Campaigns, WMS) currently offered in the WC platform. It should be easy to manage, deploy and use by non-technical people.

Phase one of this project will last 6 weeks. The scope identified above will be parsed out and assigned to each individual as part of this project. Once the scope has been assigned and phase one features have been identified, the expectation is that each developer will be responsible for ensuring their area of responsibility is completed and integrated into the larger project.

A project manager must be assigned for this initiative. Weekly status updates and communications are required.